



pytermor

Release 2.0.0-dev.17

Alexandr Shavykin

Aug 20, 2022

CONTENTS

| | | |
|----------|-----------------------------|-----------|
| 1 | Guide | 3 |
| 1.1 | Getting started | 3 |
| 1.1.1 | Installation | 3 |
| 1.1.2 | Structure | 3 |
| 1.1.3 | Features | 3 |
| 1.2 | High-level abstractions | 5 |
| 1.2.1 | Colors and Styles | 5 |
| 1.2.2 | Output format control | 5 |
| 1.2.3 | Color mode fallbacks | 5 |
| 1.2.4 | Core API | 5 |
| 1.3 | Low-level abstractions | 6 |
| 1.3.1 | Format soft reset | 6 |
| 1.3.2 | Working with Spans | 7 |
| 1.3.3 | Creating and applying SGRs | 8 |
| 1.3.4 | SGR sequence structure | 9 |
| 1.3.5 | Combining SGRs | 9 |
| 1.3.6 | Core API | 9 |
| 1.4 | Preset list | 9 |
| 1.4.1 | Meta, attributes, breakers | 10 |
| 1.4.2 | Default colors | 11 |
| 1.4.3 | Indexed colors | 12 |
| 1.5 | Color palette | 17 |
| 1.6 | Formatters and Filters | 19 |
| 1.6.1 | Auto-float formatter | 19 |
| 1.6.2 | Prefixed-unit formatter | 19 |
| 1.6.3 | Time delta formatter | 19 |
| 1.6.4 | String filters | 20 |
| 1.6.5 | Standard Library extensions | 20 |
| 2 | API reference | 21 |
| 2.1 | color | 21 |
| 2.2 | intcode | 26 |
| 2.3 | renderer | 26 |
| 2.4 | sequence | 28 |
| 2.5 | span | 31 |
| 2.6 | style | 33 |
| 2.7 | text | 34 |
| 2.8 | util | 34 |
| 2.8.1 | auto_float | 35 |
| 2.8.2 | prefixed_unit | 35 |

| | | |
|----------|----------------------------|-----------|
| 2.8.3 | time_delta | 37 |
| 2.8.4 | stdlib_ext | 39 |
| 2.8.5 | string_filter | 39 |
| 3 | Changelog | 43 |
| 4 | License | 47 |
| | Python Module Index | 49 |

(yet another) Python library designed for formatting terminal output using ANSI escape codes. Implements automatic "soft" format termination. Provides a registry of low-level SGR (Select Graphic Rendition) *sequences* and formatting *spans* (or combined sequences). Also includes a set of formatters for pretty output.

Key feature of this library is providing necessary abstractions for building complex text sections with lots of formatting, while keeping the application code clear and readable.

No dependencies besides Python Standard Library are required (*there are some for testing and docs building, though*).

@TODO This is how you **should** format examples:

We put these pieces together to create a SGR command. Thus, `ESC[1m` specifies bold (or bright) text, and `ESC[31m` specifies red foreground text. We can chain together parameters; for example, `ESC[32;47m` specifies green foreground text on a white background.

The following diagram shows a complete example for rendering the word "text" in red with a single underline.



Notes

- For terminals that support bright foreground colors, `ESC[1;3Xm` is usually equivalent to `ESC[9Xm` (where `X` is a digit in 0-7). However, the reverse does not seem to hold, at least anecdotally: `ESC[2;9Xm` usually does not render the same as `ESC[3Xm`.
- Not all terminals support every effect.

Fig. 1: <https://chrisyeh96.github.io/2020/03/28/terminal-colors.html#color-schemes>

1.1 Getting started

1.1.1 Installation

```
pip install pytermor
```

1.1.2 Structure

| Lvl | Module | Classes | Purpose |
|-----|----------------|--------------------|--|
| Hi | <i>render</i> | <i>Text</i> | Container consisting of text pieces each with attached <i>Style</i> . Renders into specified format keeping all the formatting. |
| | | <i>Style</i> | Reusable abstractions defining colors and text attributes (text color, bg color, bold attribute, underlined attribute etc). |
| | | * <i>Renderer</i> | <i>SGRRenderer</i> transforms <i>Style</i> instances into <i>Color</i> , <i>Span</i> and <i>SequenceSGR</i> instances and assembles it all up. There are several other implementations depending on required output format: <i>HtmlRenderer</i> , <i>TmuxRenderer</i> etc. |
| | <i>color</i> | <i>Color*</i> | Abstractions for color operations in different color modes (default 16-color, 256-color, RGB). Tools for color approximation and transformations. |
| Lo | ansi | <i>Span</i> | Abstraction consisting of “opening” SGR sequence defined by the developer (or taken from preset list) and complementary “closing” SGR sequence that is built automatically. |
| | | <i>SequenceSGR</i> | Abstractions for manipulating ANSI control sequences and classes-factories. |
| | <i>intcode</i> | * | Registry of escape control sequence parameters. |
| | <i>util</i> | * | Additional formatters and common methods for manipulating strings with SGRs inside. |

1.1.3 Features

One of the core concepts of the library is *Span* class. *Span* is a combination of two control sequences; it wraps specified string with pre-defined leading and trailing SGR definitions.

Example code:

```
1 from pytermor import span
2
3 print(span.RED('Feat') + span.BOLD('ures'))
```

Content-aware format nesting

Compose text spans with automatic content-aware span termination. Preset spans can safely overlap with each other (as long as they require different *breaker* sequences to reset).

```
1 from pytermor import Span
2
3 span1 = Span('blue', 'bold')
4 span2 = Span('cyan', 'inversed', 'underlined', 'italic')
5
6 msg = span1(f'Content{span2("-aware format")} nesting')
7 print(msg)
```



Flexible sequence builder

Create your own *SGR sequences* with *build()* method, which accepts color/attribute keys, integer codes and even existing *SGRs*, in any amount and in any order. Key resolving is case-insensitive.

```
1 from pytermor import sequence, build
2
3 seq1 = build('hi_blue', 1) # keys or integer codes
4 seq2 = build(seq1, sequence.ITALIC) # existing SGRs
5 seq3 = build('underlined', 'YELLOW') # case-insensitive
6
7 msg = f'{seq1}Flexible{sequence.RESET} ' + \
8       f'{seq2}sequence{sequence.RESET} ' + \
9       str(seq3) + 'builder' + str(sequence.RESET)
10 print(msg)
```


256 colors / True Color support

The library supports extended color modes:

- XTerm 256 colors indexed mode (see *Preset list*);
- True Color RGB mode (16M colors).

```

1 from pytermor import color_indexed, color_rgb, sequence
2
3 start_color = 41
4 for idx, c in enumerate(range(start_color, start_color+(36*6), 36)):
5     print(f'{color_indexed(c)}{sequence.COLOR_OFF}', end='')
6
7 print('\n')
8 for idx, c in enumerate(range(0, 256, 256//17)):
9     r = max(0, 255-c)
10    g = max(0, min(255, 127-(c*2)))
11    b = c
12    print(f'{color_rgb(r, g, b)}{sequence.COLOR_OFF}', end='')

```



Customizable output formats

@TODO

String and number formatters

@TODO

1.2 High-level abstractions

1.2.1 Colors and Styles

1.2.2 Output format control

1.2.3 Color mode fallbacks

1.2.4 Core API

@EXAMPLES

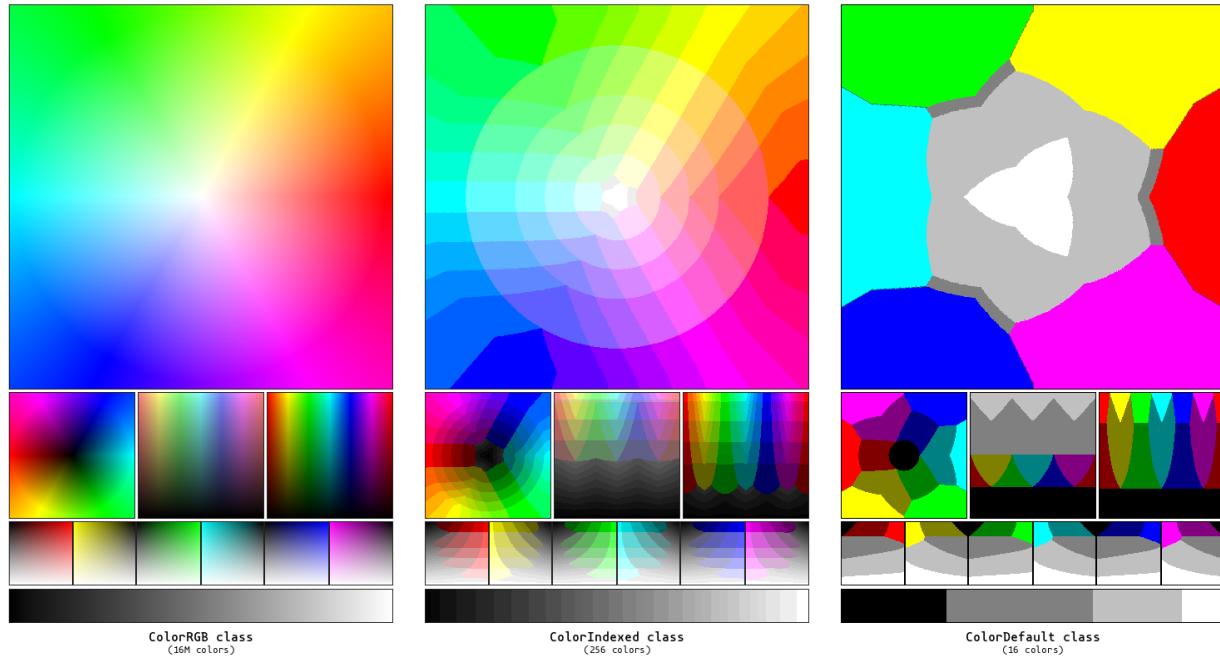


Fig. 1: Color approximations for indexed modes

1.3 Low-level abstractions

So, what's happening under the hood?

1.3.1 Format soft reset

There are two ways to manage color and attribute termination:

- hard reset (SGR-0 or `e[0m]`)
- soft reset (SGR-22, 23, 24 etc.)

The main difference between them is that *hard* reset disables all formatting after itself, while *soft* reset disables only actually necessary attributes (i.e. used as opening sequence in `Span` instance's context) and keeps the other.

That's what `Span` class is designed for: to simplify creation of soft-resetting text spans, so that developer doesn't have to restore all previously applied formats after every closing sequence.

Example

We are given a text span which is initially *bold* and *underlined*. We want to recolor a few words inside of this span. By default this will result in losing all the formatting to the right of updated text span (because `RESET`, or `e[0m`, clears all text attributes).

However, there is an option to specify what attributes should be disabled or let the library do that for you:

```
1 from pytermor import sequence, span, Span
2
3 # implicitly:
```

(continues on next page)

(continued from previous page)

```

4 span_warn = Span(93, 4)
5 # or explicitly:
6 span_warn = Span.from_seq(
7     sequence.HI_YELLOW + sequence.UNDERLINED, # sequences can be summed up, remember?
8     sequence.COLOR_OFF + sequence.UNDERLINED_OFF, # "counteractive" sequences
9     hard_reset_after=False
10 )
11
12 orig_text = span.BOLD(f'this is {sequence.BG_GRAY}the original{sequence.RESET} string')
13 updated_text = orig_text.replace('original', span_warn('updated'), 1)
14 print(orig_text, '\n', updated_text)

```



As you can see, the update went well – we kept all the previously applied formatting. Of course, this method cannot be 100% applicable; for example, imagine that original text was colored blue. After the update “string” word won’t be blue anymore, as we used `sequence.COLOR_OFF` escape sequence to neutralize our own yellow color. But it still can be helpful for a majority of cases (especially when text is generated and formatted by the same program and in one go).

1.3.2 Working with Spans

Use `Span` constructor to create new instance with specified control sequence(s) as a opening/starter sequence and **automatically composed** closing sequence that will terminate attributes defined in opening sequence while keeping the others (soft reset).

Resulting sequence params’ order is the same as argument’s order.

Each sequence param can be specified as:

- string key (see *Preset list*);
- integer param value;
- existing `SequenceSGR` instance (params will be extracted).

It’s also possible to avoid auto-composing mechanism and create `Span` with explicitly set parameters using `Span.from_seq()`.

1.3.3 Creating and applying SGRs

You can use any of predefined sequences from [sequence](#) or create your own via standard constructor. Valid argument values as well as preset constants are described in [Preset list](#) page.

There is also a set of methods for dynamic SequenceSGR creation:

- `build()` for non-specific sequences;

Important: SequenceSGR with zero params was specifically implemented to translate into an empty string and not into `e[m`, which would make sense, but also could be very entangling, as terminal emulators interpret that sequence as `e[0m`, which is *hard* reset sequence.

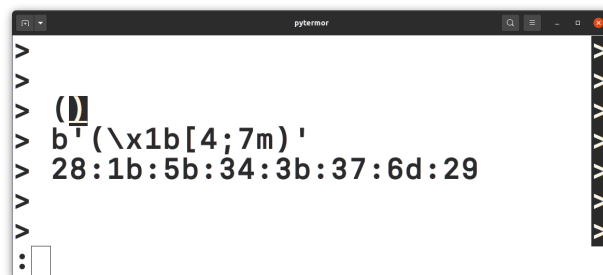
- `color_indexed()` for complex color selection sequences operating in 256-colors mode (for a complete list see [Preset list](#));
- `color_rgb()` for setting the colors in True Color 16M mode (however, some terminal emulators doesn't support it).

To get the resulting sequence chars use `encode()` method or cast instance to `str`.

```

1 from pytermor.sequence import SequenceSGR
2
3 seq = SequenceSGR(4, 7)
4 msg = f'({seq})'
5
6 print(msg + f'{SequenceSGR(0).encode()}')
7 print(str(msg.encode()))
8 print(msg.encode().hex(':'))

```



- First line is the string with encoded escape sequence;
- Second line shows up the string in raw mode, as if sequences were ignored by the terminal;
- Third line is hexademical string representation.

1.3.4 SGR sequence structure

1. `\x1b` is ESC *control character*, which opens a control sequence.
2. `[` is sequence *introducer*; it determines the type of control sequence (in this case it's CSI (Control Sequence Introducer)).
3. `4` and `7` are *parameters* of the escape sequence; they mean “underlined” and “inversed” attributes respectively. Those parameters must be separated by `;`.
4. `m` is sequence *terminator*; it also determines the sub-type of sequence, in our case SGR. Sequences of this kind are most commonly encountered.

1.3.5 Combining SGRs

One instance of `SequenceSGR` can be added to another. This will result in a new `SequenceSGR` with combined params.

```

1 from pytermor import sequence
2 from pytermor.sequence import SequenceSGR
3
4 combined = SequenceSGR(1, 31) + SequenceSGR(4)
5 print(f'{combined}{combined}{sequence.RESET}', str(combined).encode())

```

1.3.6 Core API

- `@TODO`
- `Span` constructor
- `Span.from_seq()`

1.4 Preset list

Preset lists are omitted from API docs to avoid unnecessary duplication; summary list of all presets defined in the library (not including `util.*`) is displayed here.

`@TODO` USAGE - list all memthods that accept string keys of those presets.

There are two types of color palettes used in modern terminals – first one containing 16 colors (library references that palette as *default*, see `ColorDefault`), and second one consisting of 256 colors (referenced as *indexed*, e.g. `ColorIndexed`). There is also True Color mode (referenced as *RGB* mode), but it is not palette-based.

Legend

- INT (intcode module -- 1st or 3rd SGR param value)
- SEQ (sequence module)
- SPN (span module)
- CLR (color module)
- STY (style module)

































1.4.1 Meta, attributes, breakers

| | Name | INT | SEQ | SPN | CLR | STY | Description |
|-------------------|-------------------|-----|-----|-----|-----|----------------|---|
| Meta | | | | | | | |
| | NOOP | | V | V | V | V | No-operation; always encoded as empty string |
| | RESET | 0 | V | | | | Reset all attributes and colors |
| Attributes | | | | | | | |
| | BOLD | 1 | V | V | | V ¹ | Bold or increased intensity |
| | DIM | 2 | V | V | | V | Faint, decreased intensity |
| | ITALIC | 3 | V | V | | V | Italic; <i>not widely supported</i> |
| | UNDERLINED | 4 | V | V | | V | Underline |
| | BLINK_SLOW | 5 | V | | | V ² | Set blinking to < 150 cpm |
| | BLINK_FAST | 6 | V | | | | Set blinking to 150+ cpm; <i>not widely supported</i> |
| | INVERSED | 7 | V | V | | V | Swap foreground and background colors |
| | HIDDEN | 8 | V | | | | Conceal characters; <i>not widely supported</i> |
| | CROSSLINED | 9 | V | | | V | Strikethrough |
| | DOUBLE_UNDERLINED | 21 | V | | | | Double-underline; <i>on several terminals disables BOLD instead</i> |
| | COLOR_EXTENDED | 38 | | | | | Set foreground color [<i>indexed/RGB mode</i>]; use color_indexed and color_rgb instead |
| | BG_COLOR_EXTENDED | 48 | | | | | Set background color [<i>indexed/RGB mode</i>]; use color_indexed and color_rgb instead |
| | OVERLINED | 53 | V | V | | V | Overline; <i>not widely supported</i> |
| Breakers | | | | | | | |
| | BOLD_DIM_OFF | 22 | V | | | | Disable BOLD and DIM attributes. <i>Special aspects... It's impossible to reliably disable them on a separate basis.</i> |
| | ITALIC_OFF | 23 | V | | | | Disable italic |
| | UNDERLINED_OFF | 24 | V | | | | Disable underlining |
| | BLINK_OFF | 25 | V | | | | Disable blinking |
| | INVERSED_OFF | 27 | V | | | | Disable inversing |
| | HIDDEN_OFF | 28 | V | | | | Disable concealing |
| | CROSSLINED_OFF | 29 | V | | | | Disable strikethrough |
| | COLOR_OFF | 39 | V | | | | Reset foreground color |
| | BG_COLOR_OFF | 49 | V | | | | Reset background color |
| | OVERLINED_OFF | 55 | V | | | | Disable overlining |












































¹ for this and subsequent items in “Attributes” section: as boolean flags.

² as `blink`.

1.4.2 Default colors


















































| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|---------------|-----|-----|-----|-----|-----|----------|------------|
| Foreground default colors | | | | | | | | |
|  | BLACK | 30 | V | V | V | | #000000 | Black |
|  | RED | 31 | V | V | V | | #800000 | Maroon |
|  | GREEN | 32 | V | V | V | | #008000 | Green |
|  | YELLOW | 33 | V | V | V | | #808000 | Olive |
|  | BLUE | 34 | V | V | V | | #000080 | Navy |
|  | MAGENTA | 35 | V | V | V | | #800080 | Purple |
|  | CYAN | 36 | V | V | V | | #008080 | Teal |
|  | WHITE | 37 | V | V | V | | #c0c0c0 | Silver |
| Background default colors | | | | | | | | |
|  | BG_BLACK | 40 | V | V | V | | #000000 | Black |
|  | BG_RED | 41 | V | V | V | | #800000 | Maroon |
|  | BG_GREEN | 42 | V | V | V | | #008000 | Green |
|  | BG_YELLOW | 43 | V | V | V | | #808000 | Olive |
|  | BG_BLUE | 44 | V | V | V | | #000080 | Navy |
|  | BG_MAGENTA | 45 | V | V | V | | #800080 | Purple |
|  | BG_CYAN | 46 | V | V | V | | #008080 | Teal |
|  | BG_WHITE | 47 | V | V | V | | #c0c0c0 | Silver |
| High-intensity foreground default colors | | | | | | | | |
|  | GRAY | 90 | V | V | V | | #808080 | Grey |
|  | HI_RED | 91 | V | V | V | | #ff0000 | Red |
|  | HI_GREEN | 92 | V | V | V | | #00ff00 | Lime |
|  | HI_YELLOW | 93 | V | V | V | | #ffff00 | Yellow |
|  | HI_BLUE | 94 | V | V | V | | #0000ff | Blue |
|  | HI_MAGENTA | 95 | V | V | V | | #ff00ff | Fuchsia |
|  | HI_CYAN | 96 | V | V | V | | #00ffff | Aqua |
|  | HI_WHITE | 97 | V | V | V | | #ffffff | White |
| High-intensity background default colors | | | | | | | | |
|  | BG_GRAY | 100 | V | V | V | | #808080 | Grey |
|  | BG_HI_RED | 101 | V | V | V | | #ff0000 | Red |
|  | BG_HI_GREEN | 102 | V | V | V | | #00ff00 | Lime |
|  | BG_HI_YELLOW | 103 | V | V | V | | #ffff00 | Yellow |
|  | BG_HI_BLUE | 104 | V | V | V | | #0000ff | Blue |
|  | BG_HI_MAGENTA | 105 | V | V | V | | #ff00ff | Fuchsia |
|  | BG_HI_CYAN | 106 | V | V | V | | #00ffff | Aqua |
|  | BG_HI_WHITE | 107 | V | V | V | | #ffffff | White |

1.4.3 Indexed colors

| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|--------------------------|-----|-----|-----|-----|-----|----------|---------------------|
|  | XTERM_BLACK ³ | 0 | | | V | | #000000 | |
|  | XTERM_MAROON | 1 | | | V | | #800000 | |
|  | XTERM_GREEN | 2 | | | V | | #008000 | |
|  | XTERM_OLIVE | 3 | | | V | | #808000 | |
|  | XTERM_NAVY | 4 | | | V | | #000080 | |
|  | XTERM_PURPLE_5 | 5 | | | V | | #800080 | Purple ⁴ |
|  | XTERM_TEAL | 6 | | | V | | #008080 | |
|  | XTERM_SILVER | 7 | | | V | | #c0c0c0 | |
|  | XTERM_GREY | 8 | | | V | | #808080 | |
|  | XTERM_RED | 9 | | | V | | #ff0000 | |
|  | XTERM_LIME | 10 | | | V | | #00ff00 | |
|  | XTERM_YELLOW | 11 | | | V | | #ffff00 | |
|  | XTERM_BLUE | 12 | | | V | | #0000ff | |
|  | XTERM_FUCHSIA | 13 | | | V | | #ff00ff | |
|  | XTERM_AQUA | 14 | | | V | | #00ffff | |
| | XTERM_WHITE | 15 | | | V | | #ffffff | |
|  | XTERM_GREY_0 | 16 | | | V | | #000000 | |
|  | XTERM_NAVY_BLUE | 17 | | | V | | #00005f | |
|  | XTERM_DARK_BLUE | 18 | | | V | | #000087 | |
|  | XTERM_BLUE_3 | 19 | | | V | | #0000af | |
|  | XTERM_BLUE_2 | 20 | | | V | | #0000d7 | Blue3 |
|  | XTERM_BLUE_1 | 21 | | | V | | #0000ff | |
|  | XTERM_DARK_GREEN | 22 | | | V | | #005f00 | |
|  | XTERM_DEEP_SKY_BLUE_7 | 23 | | | V | | #005f5f | DeepSkyBlue4 |
|  | XTERM_DEEP_SKY_BLUE_6 | 24 | | | V | | #005f87 | DeepSkyBlue4 |
|  | XTERM_DEEP_SKY_BLUE_5 | 25 | | | V | | #005faf | DeepSkyBlue4 |
|  | XTERM_DODGER_BLUE_3 | 26 | | | V | | #005fd7 | |
|  | XTERM_DODGER_BLUE_2 | 27 | | | V | | #005fff | |
|  | XTERM_GREEN_5 | 28 | | | V | | #008700 | Green4 |
|  | XTERM_SPRING_GREEN_4 | 29 | | | V | | #00875f | |
|  | XTERM_TURQUOISE_4 | 30 | | | V | | #008787 | |
|  | XTERM_DEEP_SKY_BLUE_4 | 31 | | | V | | #0087af | DeepSkyBlue3 |
|  | XTERM_DEEP_SKY_BLUE_3 | 32 | | | V | | #0087d7 | |
|  | XTERM_DODGER_BLUE_1 | 33 | | | V | | #0087ff | |
|  | XTERM_GREEN_4 | 34 | | | V | | #00af00 | Green3 |
|  | XTERM_SPRING_GREEN_5 | 35 | | | V | | #00af5f | SpringGreen3 |
|  | XTERM_DARK_CYAN | 36 | | | V | | #00af87 | |
|  | XTERM_LIGHT_SEA_GREEN | 37 | | | V | | #00afaf | |
|  | XTERM_DEEP_SKY_BLUE_2 | 38 | | | V | | #00afd7 | |
|  | XTERM_DEEP_SKY_BLUE_1 | 39 | | | V | | #00afff | |
|  | XTERM_GREEN_3 | 40 | | | V | | #00d700 | |
|  | XTERM_SPRING_GREEN_3 | 41 | | | V | | #00d75f | |
|  | XTERM_SPRING_GREEN_6 | 42 | | | V | | #00d787 | SpringGreen2 |
|  | XTERM_CYAN_3 | 43 | | | V | | #00d7af | |
|  | XTERM_DARK_TURQUOISE | 44 | | | V | | #00d7d7 | |
|  | XTERM_TURQUOISE_2 | 45 | | | V | | #00d7ff | |
|  | XTERM_GREEN_2 | 46 | | | V | | #00ff00 | Green1 |


















continues on next page

Table 2 – continued from previous page

| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|---------------------------|-----|-----|-----|-----|-----|----------|---------------|
|  | XTERM_SPRING_GREEN_2 | 47 | | | V | | #00ff5f | |
|  | XTERM_SPRING_GREEN_1 | 48 | | | V | | #00ff87 | |
|  | XTERM_MEDIUM_SPRING_GREEN | 49 | | | V | | #00ffa7 | |
|  | XTERM_CYAN_2 | 50 | | | V | | #00ffd7 | |
|  | XTERM_CYAN_1 | 51 | | | V | | #00ffff | |
|  | XTERM_DARK_RED_2 | 52 | | | V | | #5f0000 | DarkRed |
|  | XTERM_DEEP_PINK_8 | 53 | | | V | | #5f005f | DeepPink4 |
|  | XTERM_PURPLE_6 | 54 | | | V | | #5f0087 | Purple4 |
|  | XTERM_PURPLE_4 | 55 | | | V | | #5f00af | |
|  | XTERM_PURPLE_3 | 56 | | | V | | #5f00d7 | |
|  | XTERM_BLUE_VIOLET | 57 | | | V | | #5f00ff | |
|  | XTERM_ORANGE_4 | 58 | | | V | | #5f5f00 | |
|  | XTERM_GREY_37 | 59 | | | V | | #5f5f5f | |
|  | XTERM_MEDIUM_PURPLE_7 | 60 | | | V | | #5f5f87 | MediumPurple4 |
|  | XTERM_SLATE_BLUE_3 | 61 | | | V | | #5f5faf | |
|  | XTERM_SLATE_BLUE_2 | 62 | | | V | | #5f5fd7 | SlateBlue3 |
|  | XTERM_ROYAL_BLUE_1 | 63 | | | V | | #5f5fff | |
|  | XTERM_CHARTREUSE_6 | 64 | | | V | | #5f8700 | Chartreuse4 |
|  | XTERM_DARK_SEA_GREEN_9 | 65 | | | V | | #5f875f | DarkSeaGreen4 |
|  | XTERM_PALE_TURQUOISE_4 | 66 | | | V | | #5f8787 | |
|  | XTERM_STEEL_BLUE | 67 | | | V | | #5f87af | |
|  | XTERM_STEEL_BLUE_3 | 68 | | | V | | #5f87d7 | |
|  | XTERM_CORNFLOWER_BLUE | 69 | | | V | | #5f87ff | |
|  | XTERM_CHARTREUSE_5 | 70 | | | V | | #5faf00 | Chartreuse3 |
|  | XTERM_DARK_SEA_GREEN_8 | 71 | | | V | | #5faf5f | DarkSeaGreen4 |
|  | XTERM_CADET_BLUE_2 | 72 | | | V | | #5faf87 | CadetBlue |
|  | XTERM_CADET_BLUE | 73 | | | V | | #5fafaf | |
|  | XTERM_SKY_BLUE_3 | 74 | | | V | | #5fafd7 | |
|  | XTERM_STEEL_BLUE_2 | 75 | | | V | | #5fafff | SteelBlue1 |
|  | XTERM_CHARTREUSE_4 | 76 | | | V | | #5fd700 | Chartreuse3 |
|  | XTERM_PALE_GREEN_4 | 77 | | | V | | #5fd75f | PaleGreen3 |
|  | XTERM_SEA_GREEN_3 | 78 | | | V | | #5fd787 | |
|  | XTERM_AQUAMARINE_3 | 79 | | | V | | #5fd7af | |
|  | XTERM_MEDIUM_TURQUOISE | 80 | | | V | | #5fd7d7 | |
|  | XTERM_STEEL_BLUE_1 | 81 | | | V | | #5fd7ff | |
|  | XTERM_CHARTREUSE_2 | 82 | | | V | | #5fff00 | |
|  | XTERM_SEA_GREEN_4 | 83 | | | V | | #5fff5f | SeaGreen2 |
|  | XTERM_SEA_GREEN_2 | 84 | | | V | | #5fff87 | SeaGreen1 |
|  | XTERM_SEA_GREEN_1 | 85 | | | V | | #5fffaf | |
|  | XTERM_AQUAMARINE_2 | 86 | | | V | | #5fffd7 | Aquamarine1 |
|  | XTERM_DARK_SLATE_GRAY_2 | 87 | | | V | | #5ffffff | |
|  | XTERM_DARK_RED | 88 | | | V | | #870000 | |
|  | XTERM_DEEP_PINK_7 | 89 | | | V | | #87005f | DeepPink4 |
|  | XTERM_DARK_MAGENTA_2 | 90 | | | V | | #870087 | DarkMagenta |
|  | XTERM_DARK_MAGENTA | 91 | | | V | | #8700af | |
|  | XTERM_DARK_VIOLET_2 | 92 | | | V | | #8700d7 | DarkViolet |
|  | XTERM_PURPLE_2 | 93 | | | V | | #8700ff | Purple |
|  | XTERM_ORANGE_3 | 94 | | | V | | #875f00 | Orange4 |
|  | XTERM_LIGHT_PINK_3 | 95 | | | V | | #875f5f | LightPink4 |


















































continues on next page

Table 2 – continued from previous page

| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|--------------------------|-----|-----|-----|-----|-----|----------|-----------------|
|  | XTERM_PLUM_4 | 96 | | | V | | #875f87 | |
|  | XTERM_MEDIUM_PURPLE_6 | 97 | | | V | | #875faf | MediumPurple3 |
|  | XTERM_MEDIUM_PURPLE_5 | 98 | | | V | | #875fd7 | MediumPurple3 |
|  | XTERM_SLATE_BLUE_1 | 99 | | | V | | #875fff | |
|  | XTERM_YELLOW_6 | 100 | | | V | | #878700 | Yellow4 |
|  | XTERM_WHEAT_4 | 101 | | | V | | #87875f | |
|  | XTERM_GREY_53 | 102 | | | V | | #878787 | |
|  | XTERM_LIGHT_SLATE_GREY | 103 | | | V | | #8787af | |
|  | XTERM_MEDIUM_PURPLE_4 | 104 | | | V | | #8787d7 | MediumPurple |
|  | XTERM_LIGHT_SLATE_BLUE | 105 | | | V | | #8787ff | |
|  | XTERM_YELLOW_4 | 106 | | | V | | #87af00 | |
|  | XTERM_DARK_OLIVE_GREEN_6 | 107 | | | V | | #87af5f | DarkOliveGreen3 |
|  | XTERM_DARK_SEA_GREEN_7 | 108 | | | V | | #87af87 | DarkSeaGreen |
|  | XTERM_LIGHT_SKY_BLUE_3 | 109 | | | V | | #87afaf | |
|  | XTERM_LIGHT_SKY_BLUE_2 | 110 | | | V | | #87afd7 | LightSkyBlue3 |
|  | XTERM_SKY_BLUE_2 | 111 | | | V | | #87afff | |
|  | XTERM_CHARTREUSE_3 | 112 | | | V | | #87d700 | Chartreuse2 |
|  | XTERM_DARK_OLIVE_GREEN_4 | 113 | | | V | | #87d75f | DarkOliveGreen3 |
|  | XTERM_PALE_GREEN_3 | 114 | | | V | | #87d787 | |
|  | XTERM_DARK_SEA_GREEN_5 | 115 | | | V | | #87d7af | DarkSeaGreen3 |
|  | XTERM_DARK_SLATE_GRAY_3 | 116 | | | V | | #87d7d7 | |
|  | XTERM_SKY_BLUE_1 | 117 | | | V | | #87d7ff | |
|  | XTERM_CHARTREUSE_1 | 118 | | | V | | #87ff00 | |
|  | XTERM_LIGHT_GREEN_2 | 119 | | | V | | #87ff5f | LightGreen |
|  | XTERM_LIGHT_GREEN | 120 | | | V | | #87ff87 | |
|  | XTERM_PALE_GREEN_1 | 121 | | | V | | #87ffaf | |
|  | XTERM_AQUAMARINE_1 | 122 | | | V | | #87ffd7 | |
|  | XTERM_DARK_SLATE_GRAY_1 | 123 | | | V | | #87ffff | |
|  | XTERM_RED_4 | 124 | | | V | | #af0000 | Red3 |
|  | XTERM_DEEP_PINK_6 | 125 | | | V | | #af005f | DeepPink4 |
|  | XTERM_MEDIUM_VIOLET_RED | 126 | | | V | | #af0087 | |
|  | XTERM_MAGENTA_6 | 127 | | | V | | #af00af | Magenta3 |
|  | XTERM_DARK_VIOLET | 128 | | | V | | #af00d7 | |
|  | XTERM_PURPLE | 129 | | | V | | #af00ff | |
|  | XTERM_DARK_ORANGE_3 | 130 | | | V | | #af5f00 | |
|  | XTERM_INDIAN_RED_4 | 131 | | | V | | #af5f5f | IndianRed |
|  | XTERM_HOT_PINK_5 | 132 | | | V | | #af5f87 | HotPink3 |
|  | XTERM_MEDIUM_ORCHID_4 | 133 | | | V | | #af5faf | MediumOrchid3 |
|  | XTERM_MEDIUM_ORCHID_3 | 134 | | | V | | #af5fd7 | MediumOrchid |
|  | XTERM_MEDIUM_PURPLE_2 | 135 | | | V | | #af5fff | |
|  | XTERM_DARK_GOLDENROD | 136 | | | V | | #af8700 | |
|  | XTERM_LIGHT_SALMON_3 | 137 | | | V | | #af875f | |
|  | XTERM_ROSY_BROWN | 138 | | | V | | #af8787 | |
|  | XTERM_GREY_63 | 139 | | | V | | #af87af | |
|  | XTERM_MEDIUM_PURPLE_3 | 140 | | | V | | #af87d7 | MediumPurple2 |
|  | XTERM_MEDIUM_PURPLE_1 | 141 | | | V | | #af87ff | |
|  | XTERM_GOLD_3 | 142 | | | V | | #afaf00 | |
|  | XTERM_DARK_KHAKI | 143 | | | V | | #afaf5f | |
|  | XTERM_NAVAJO_WHITE_3 | 144 | | | V | | #afaf87 | |














continues on next page

Table 2 – continued from previous page

| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|--------------------------|-----|-----|-----|-----|-----|----------|-----------------|
|  | XTERM_GREY_69 | 145 | | | V | | #afafaf | |
|  | XTERM_LIGHT_STEEL_BLUE_3 | 146 | | | V | | #afafd7 | |
|  | XTERM_LIGHT_STEEL_BLUE_2 | 147 | | | V | | #afafff | LightSteelBlue |
|  | XTERM_YELLOW_5 | 148 | | | V | | #afd700 | Yellow3 |
|  | XTERM_DARK_OLIVE_GREEN_5 | 149 | | | V | | #afd75f | DarkOliveGreen3 |
|  | XTERM_DARK_SEA_GREEN_6 | 150 | | | V | | #afd787 | DarkSeaGreen3 |
|  | XTERM_DARK_SEA_GREEN_4 | 151 | | | V | | #afd7af | DarkSeaGreen2 |
|  | XTERM_LIGHT_CYAN_3 | 152 | | | V | | #afd7d7 | |
|  | XTERM_LIGHT_SKY_BLUE_1 | 153 | | | V | | #afd7ff | |
|  | XTERM_GREEN_YELLOW | 154 | | | V | | #afff00 | |
|  | XTERM_DARK_OLIVE_GREEN_3 | 155 | | | V | | #afff5f | DarkOliveGreen2 |
|  | XTERM_PALE_GREEN_2 | 156 | | | V | | #afff87 | PaleGreen1 |
|  | XTERM_DARK_SEA_GREEN_3 | 157 | | | V | | #afffaf | DarkSeaGreen2 |
|  | XTERM_DARK_SEA_GREEN_1 | 158 | | | V | | #afffd7 | |
|  | XTERM_PALE_TURQUOISE_1 | 159 | | | V | | #afffff | |
|  | XTERM_RED_3 | 160 | | | V | | #d70000 | |
|  | XTERM_DEEP_PINK_5 | 161 | | | V | | #d7005f | DeepPink3 |
|  | XTERM_DEEP_PINK_3 | 162 | | | V | | #d70087 | |
|  | XTERM_MAGENTA_3 | 163 | | | V | | #d700af | |
|  | XTERM_MAGENTA_5 | 164 | | | V | | #d700d7 | Magenta3 |
|  | XTERM_MAGENTA_4 | 165 | | | V | | #d700ff | Magenta2 |
|  | XTERM_DARK_ORANGE_2 | 166 | | | V | | #d75f00 | DarkOrange3 |
|  | XTERM_INDIAN_RED_3 | 167 | | | V | | #d75f5f | IndianRed |
|  | XTERM_HOT_PINK_4 | 168 | | | V | | #d75f87 | HotPink3 |
|  | XTERM_HOT_PINK_3 | 169 | | | V | | #d75faf | HotPink2 |
|  | XTERM_ORCHID_3 | 170 | | | V | | #d75fd7 | Orchid |
|  | XTERM_MEDIUM_ORCHID_2 | 171 | | | V | | #d75fff | MediumOrchid1 |
|  | XTERM_ORANGE_2 | 172 | | | V | | #d78700 | Orange3 |
|  | XTERM_LIGHT_SALMON_2 | 173 | | | V | | #d7875f | LightSalmon3 |
|  | XTERM_LIGHT_PINK_2 | 174 | | | V | | #d78787 | LightPink3 |
|  | XTERM_PINK_3 | 175 | | | V | | #d787af | |
|  | XTERM_PLUM_3 | 176 | | | V | | #d787d7 | |
|  | XTERM_VIOLET | 177 | | | V | | #d787ff | |
|  | XTERM_GOLD_2 | 178 | | | V | | #d7af00 | Gold3 |
|  | XTERM_LIGHT_GOLDENROD_5 | 179 | | | V | | #d7af5f | LightGoldenrod3 |
|  | XTERM_TAN | 180 | | | V | | #d7af87 | |
|  | XTERM_MISTY_ROSE_3 | 181 | | | V | | #d7afaf | |
|  | XTERM_THISTLE_3 | 182 | | | V | | #d7afd7 | |
|  | XTERM_PLUM_2 | 183 | | | V | | #d7afff | |
|  | XTERM_YELLOW_3 | 184 | | | V | | #d7d700 | |
|  | XTERM_KHAKI_3 | 185 | | | V | | #d7d75f | |
|  | XTERM_LIGHT_GOLDENROD_3 | 186 | | | V | | #d7d787 | LightGoldenrod2 |
|  | XTERM_LIGHT_YELLOW_3 | 187 | | | V | | #d7d7af | |
|  | XTERM_GREY_84 | 188 | | | V | | #d7d7d7 | |
|  | XTERM_LIGHT_STEEL_BLUE_1 | 189 | | | V | | #d7d7ff | |
|  | XTERM_YELLOW_2 | 190 | | | V | | #d7ff00 | |
|  | XTERM_DARK_OLIVE_GREEN_2 | 191 | | | V | | #d7ff5f | DarkOliveGreen1 |
|  | XTERM_DARK_OLIVE_GREEN_1 | 192 | | | V | | #d7ff87 | |
|  | XTERM_DARK_SEA_GREEN_2 | 193 | | | V | | #d7ffaf | DarkSeaGreen1 |

continues on next page

Table 2 – continued from previous page

| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|-------------------------|-----|-----|-----|-----|-----|----------|-----------------|
|  | XTERM_HONEYDEW_2 | 194 | | | V | | #d7ffd7 | |
|  | XTERM_LIGHT_CYAN_1 | 195 | | | V | | #d7ffff | |
|  | XTERM_RED_1 | 196 | | | V | | #ff0000 | |
|  | XTERM_DEEP_PINK_4 | 197 | | | V | | #ff005f | DeepPink2 |
|  | XTERM_DEEP_PINK_2 | 198 | | | V | | #ff0087 | DeepPink1 |
|  | XTERM_DEEP_PINK_1 | 199 | | | V | | #ff00af | |
|  | XTERM_MAGENTA_2 | 200 | | | V | | #ff00d7 | |
|  | XTERM_MAGENTA_1 | 201 | | | V | | #ff00ff | |
|  | XTERM_ORANGE_RED_1 | 202 | | | V | | #ff5f00 | |
|  | XTERM_INDIAN_RED_1 | 203 | | | V | | #ff5f5f | |
|  | XTERM_INDIAN_RED_2 | 204 | | | V | | #ff5f87 | IndianRed1 |
|  | XTERM_HOT_PINK_2 | 205 | | | V | | #ff5faf | HotPink |
|  | XTERM_HOT_PINK | 206 | | | V | | #ff5fd7 | |
|  | XTERM_MEDIUM_ORCHID_1 | 207 | | | V | | #ff5fff | |
|  | XTERM_DARK_ORANGE | 208 | | | V | | #ff8700 | |
|  | XTERM_SALMON_1 | 209 | | | V | | #ff875f | |
|  | XTERM_LIGHT_CORAL | 210 | | | V | | #ff8787 | |
|  | XTERM_PALE_VIOLET_RED_1 | 211 | | | V | | #ff87af | |
|  | XTERM_ORCHID_2 | 212 | | | V | | #ff87d7 | |
|  | XTERM_ORCHID_1 | 213 | | | V | | #ff87ff | |
|  | XTERM_ORANGE_1 | 214 | | | V | | #ffaaf00 | |
|  | XTERM_SANDY_BROWN | 215 | | | V | | #ffaaf5f | |
|  | XTERM_LIGHT_SALMON_1 | 216 | | | V | | #ffaaf87 | |
|  | XTERM_LIGHT_PINK_1 | 217 | | | V | | #ffaafaf | |
|  | XTERM_PINK_1 | 218 | | | V | | #ffaafd7 | |
|  | XTERM_PLUM_1 | 219 | | | V | | #ffaafff | |
|  | XTERM_GOLD_1 | 220 | | | V | | #ffd700 | |
|  | XTERM_LIGHT_GOLDENROD_4 | 221 | | | V | | #ffd75f | LightGoldenrod2 |
|  | XTERM_LIGHT_GOLDENROD_2 | 222 | | | V | | #ffd787 | |
|  | XTERM_NAVAJO_WHITE_1 | 223 | | | V | | #ffd7af | |
|  | XTERM_MISTY_ROSE_1 | 224 | | | V | | #ffd7d7 | |
|  | XTERM_THISTLE_1 | 225 | | | V | | #ffd7ff | |
|  | XTERM_YELLOW_1 | 226 | | | V | | #ffff00 | |
|  | XTERM_LIGHT_GOLDENROD_1 | 227 | | | V | | #ffff5f | |
|  | XTERM_KHAKI_1 | 228 | | | V | | #ffff87 | |
|  | XTERM_WHEAT_1 | 229 | | | V | | #ffffaf | |
|  | XTERM_CORNSILK_1 | 230 | | | V | | #ffffd7 | |
|  | XTERM_GREY_100 | 231 | | | V | | #ffffff | |
|  | XTERM_GREY_3 | 232 | | | V | | #080808 | |
|  | XTERM_GREY_7 | 233 | | | V | | #121212 | |
|  | XTERM_GREY_11 | 234 | | | V | | #1c1c1c | |
|  | XTERM_GREY_15 | 235 | | | V | | #262626 | |
|  | XTERM_GREY_19 | 236 | | | V | | #303030 | |
|  | XTERM_GREY_23 | 237 | | | V | | #3a3a3a | |
|  | XTERM_GREY_27 | 238 | | | V | | #444444 | |
|  | XTERM_GREY_30 | 239 | | | V | | #4e4e4e | |
|  | XTERM_GREY_35 | 240 | | | V | | #585858 | |
|  | XTERM_GREY_39 | 241 | | | V | | #626262 | |
|  | XTERM_GREY_42 | 242 | | | V | | #6c6c6c | |

continues on next page

Table 2 – continued from previous page

| | Name | INT | SEQ | SPN | CLR | STY | RGB code | XTerm name |
|---|---------------|-----|-----|-----|-----|-----|----------|------------|
| ■ | XTERM_GREY_46 | 243 | | | V | | #767676 | |
| ■ | XTERM_GREY_50 | 244 | | | V | | #808080 | |
| ■ | XTERM_GREY_54 | 245 | | | V | | #8a8a8a | |
| ■ | XTERM_GREY_58 | 246 | | | V | | #949494 | |
| ■ | XTERM_GREY_62 | 247 | | | V | | #9e9e9e | |
| ■ | XTERM_GREY_66 | 248 | | | V | | #a8a8a8 | |
| ■ | XTERM_GREY_70 | 249 | | | V | | #b2b2b2 | |
| ■ | XTERM_GREY_74 | 250 | | | V | | #bcbcbc | |
| ■ | XTERM_GREY_78 | 251 | | | V | | #c6c6c6 | |
| ■ | XTERM_GREY_82 | 252 | | | V | | #d0d0d0 | |
| ■ | XTERM_GREY_85 | 253 | | | V | | #dadada | |
| ■ | XTERM_GREY_89 | 254 | | | V | | #e4e4e4 | |
| ■ | XTERM_GREY_93 | 255 | | | V | | #eeeeee | |

Sources

1. https://en.wikipedia.org/wiki/ANSI_escape_code
2. <https://www.ditig.com/256-colors-cheat-sheet>

1.5 Color palette

Actual colors of *default* palette depend on user's terminal settings, i.e. the result color of `ColorDefault` is not guaranteed to exactly match the corresponding color listed below. What's more, note that *default* palette is actually a part of *indexed* one (first 16 colors of 256-color table).

Important: @TODO The approximation algorithm was explicitly made to ignore these colors because otherwise the results of transforming *RGB* values into *indexed* ones would be unpredictable, in addition to different results for different users, depending on their terminal emulator setup.

However, it doesn't mean that `ColorDefault` is useless. Just the opposite – it's ideal for situations when you don't actually **have to** set exact values and it's easier to specify estimation of desired color. I.e. setting color to 'red' is usually more than enough for displaying an error message – we don't really care of precise hue or brightness values for it.

Approximation algorithm is as simple as iterating through all colors in the *lookup table* (which contains all possible ... @TODO finish

³ First 16 colors are effectively the same as colors in *default* 16-color mode and share with them the same color values (and depend on terminal color scheme as well).

⁴ XTerm name list contains duplicates; variable names for these were slightly modified (different numbers at the end) to avoid namespace conflicts. Every changed name is displayed with **bold** font.

| | | | | | | | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| | 000 #000000 | 001 #800000 | 002 #008000 | 003 #808000 | 004 #000080 | 005 #800080 | 006 #008080 | 007 #c0c0c0 | | | |
| | 008 #808080 | 009 #ff0000 | 010 #00ff00 | 011 #ffff00 | 012 #0000ff | 013 #ff00ff | 014 #00ffff | 015 #ffffff | | | |
| 016 #000000 | 022 #005f00 | 028 #008700 | 034 #00af00 | 040 #00d700 | 046 #00ff00 | 082 #5fff00 | 076 #5fd700 | 070 #5faf00 | 064 #5f8700 | 058 #5f5f00 | 052 #5f0000 |
| 017 #00005f | 023 #005f5f | 029 #00875f | 035 #00af5f | 041 #00d75f | 047 #00ff5f | 083 #5fff5f | 077 #5fd75f | 071 #5faf5f | 065 #5f875f | 059 #5f5f5f | 053 #5f005f |
| 018 #000087 | 024 #005f87 | 030 #008787 | 036 #00af87 | 042 #00d787 | 048 #00ff87 | 084 #5fff87 | 078 #5fd787 | 072 #5faf87 | 066 #5f8787 | 060 #5f5f87 | 054 #5f0087 |
| 019 #0000af | 025 #005faf | 031 #0087af | 037 #00afaf | 043 #00d7af | 049 #00ffaf | 085 #5fffaf | 079 #5fd7af | 073 #5fafaf | 067 #5f87af | 061 #5f5faf | 055 #5f00af |
| 020 #0000d7 | 026 #005fd7 | 032 #0087d7 | 038 #00afd7 | 044 #00dd7 | 050 #00ffd7 | 086 #5fffd7 | 080 #5fd7d7 | 074 #5fadd7 | 068 #5f87d7 | 062 #5f5fd7 | 056 #5f00d7 |
| 021 #0000ff | 027 #005fff | 033 #0087ff | 039 #00afff | 045 #00d7ff | 051 #00ffff | 087 #5fffff | 081 #5fd7ff | 075 #5fafff | 069 #5f87ff | 063 #5f5fff | 057 #5f00ff |
| 093 #8700ff | 099 #875fff | 105 #8787ff | 111 #87afff | 117 #87d7ff | 123 #87ffff | 159 #afffff | 153 #afd7ff | 147 #afafff | 141 #af87ff | 135 #af5fff | 129 #af00ff |
| 092 #8700d7 | 098 #875fd7 | 104 #8787d7 | 110 #87afd7 | 116 #87dd7 | 122 #87ffd7 | 158 #afffd7 | 152 #afd7d7 | 146 #afadd7 | 140 #af87d7 | 134 #af5fd7 | 128 #af00d7 |
| 091 #8700af | 097 #875faf | 103 #8787af | 109 #87afaf | 115 #87d7af | 121 #87ffaf | 157 #afffaf | 151 #afd7af | 145 #afafaf | 139 #af87af | 133 #af5faf | 127 #af00af |
| 090 #870087 | 096 #875f87 | 102 #878787 | 108 #87af87 | 114 #87d787 | 120 #87ff87 | 156 #afff87 | 150 #afd787 | 144 #afaf87 | 138 #af8787 | 132 #af5f87 | 126 #af0087 |
| 089 #87005f | 095 #875f5f | 101 #87875f | 107 #87af5f | 113 #87d75f | 119 #87ff5f | 155 #afff5f | 149 #afd75f | 143 #afaf5f | 137 #af875f | 131 #af5f5f | 125 #af005f |
| 088 #870000 | 094 #875f00 | 100 #878700 | 106 #87af00 | 112 #87d700 | 118 #87ff00 | 154 #afff00 | 148 #afd700 | 142 #afaf00 | 136 #af8700 | 130 #af5f00 | 124 #af0000 |
| 160 #d70000 | 166 #d75f00 | 172 #d78700 | 178 #dfa00 | 184 #dfd00 | 190 #dff00 | 226 #ffff00 | 220 #ffd00 | 214 #ffa00 | 208 #ff8700 | 202 #ff5f00 | 196 #ff0000 |
| 161 #d7005f | 167 #d75f5f | 173 #d7875f | 179 #dfa5f | 185 #dfd5f | 191 #dff5f | 227 #ffff5f | 221 #ffd5f | 215 #ffa5f | 209 #ff875f | 203 #ff5f5f | 197 #ff005f |
| 162 #d70087 | 168 #d75f87 | 174 #d78787 | 180 #dfa87 | 186 #dfd87 | 192 #dff87 | 228 #ffff87 | 222 #ffd87 | 216 #ffa87 | 210 #ff8787 | 204 #ff5f87 | 198 #ff0087 |
| 163 #d700af | 169 #d75faf | 175 #d787af | 181 #dfaaf | 187 #dfdaf | 193 #dffaf | 229 #ffffaf | 223 #ffdaf | 217 #ffaaf | 211 #ff87af | 205 #ff5faf | 199 #ff00af |
| 164 #d700d7 | 170 #d75fd7 | 176 #d787d7 | 182 #dafdf | 188 #dfd7d7 | 194 #dff7d7 | 230 #ffffd7 | 224 #ffd7d7 | 218 #ffd7d7 | 212 #ff87d7 | 206 #ff5fd7 | 200 #ff00d7 |
| 165 #d700ff | 171 #d75fff | 177 #d787ff | 183 #dafff | 189 #dff7ff | 195 #dff7ff | 231 #ffffff | 225 #ffd7ff | 219 #ffa7ff | 213 #ff87ff | 207 #ff5fff | 201 #ff00ff |
| 232 #080808 | 233 #121212 | 234 #1c1c1c | 235 #262626 | 236 #303030 | 237 #3a3a3a | 238 #444444 | 239 #4e4e4e | 240 #585858 | 241 #626262 | 242 #6c6c6c | 243 #767676 |
| 244 #808080 | 245 #8a8a8a | 246 #949494 | 247 #9e9e9e | 248 #a8a8a8 | 249 #b2b2b2 | 250 #bcbcbc | 251 #c6c6c6 | 252 #d0d0d0 | 253 #dadada | 254 #e4e4e4 | 255 #eeeeee |

Fig. 2: Indexed mode palette

Sources

1. <https://www.tweaking4all.com/software/linux-software/xterm-color-cheat-sheet/>

1.6 Formatters and Filters

The library contains @TODO

1.6.1 Auto-float formatter

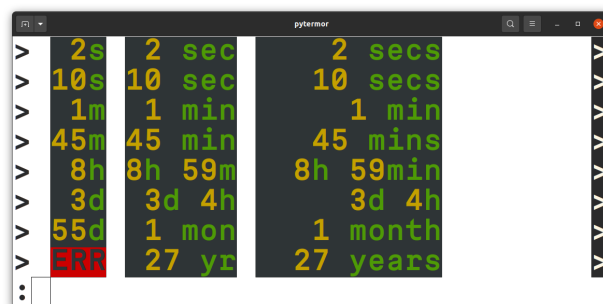
1.6.2 Prefixed-unit formatter

1.6.3 Time delta formatter

```

1 from pytermor import Style, color, ColorRGB
2 from pytermor import renderer
3 from pytermor.renderer import RendererManager, SGRRenderer
4 from pytermor.util import time_delta
5
6 seconds_list = [2, 10, 60, 2700, 32340, 273600, 4752000, 864000000]
7 max_len_list = [3, 6, 10]
8
9 custom_stylesheet = time_delta.TimeDeltaStylesheet(
10     default=Style(bg=0x202028),
11     digit=Style(0x3333000, 'yellow'),
12     unit=Style(fg='green', bg=0x202028, underlined=True),
13     overflow=Style(fg=color.BLACK, bg='hi_red', bold=True),
14 )
15 for max_len in max_len_list:
16     formatter = time_delta.registry.find_matching(max_len)
17     formatter.stylesheet = custom_stylesheet
18
19 RendererManager.set_up(SGRRenderer)
20 for seconds in seconds_list:
21     for max_len in max_len_list:
22         formatter = time_delta.registry.get_by_max_len(max_len)
23         print(formatter.format(seconds, True), end=' ')
24 print()

```



1.6.4 String filters

1.6.5 Standard Library extensions

@TODO

API REFERENCE

2.1 color

@TODO

@TODO black/white text selection depending on bg

class pytermor.color.Color

Abstract superclass for other Colors.

__init__(*hex_value: Optional[int] = None, use_for_approximations: bool = True*)

static hex_value_to_hsv_channels(*hex_value: int*) → Tuple[int, float, float]

Transforms *hex_value* in 0xffffffff format into tuple of three numbers corresponding to *hue*, *saturation* and *value* channel values respectively. *Hue* is within [0, 359] range, *saturation* and *value* are within [0; 1] range.

static hex_value_to_rgb_channels(*hex_value: int*) → Tuple[int, int, int]

Transforms *hex_value* in 0xffffffff format into tuple of three integers corresponding to *red*, *blue* and *green* channel value respectively. Values are within [0; 255] range.

```
>>> Color.hex_value_to_rgb_channels(0x80ff80)
(128, 255, 128)
>>> Color.hex_value_to_rgb_channels(0x000001)
(0, 0, 1)
```

abstract classmethod get_default() → *Color*

Returns

Fallback instance of *Color* inheritor (if registries are empty).

abstract classmethod find_closest(*hex_value: int*) → *Color*

Wrapper for *Approximator.find_closest()*.

Parameters

hex_value – Integer color value in 0xffffffff format.

Returns

Nearest found color of specified type.

abstract to_sgr(*bg: bool = False*) → *SequenceSGR*

property hex_value: int | None

format_value(*prefix: str | None = '0x', noop_str: str = '^'*) → str

class pytermor.color.ColorDefault

Bases: [Color](#)

__init__(*hex_value: int, code_fg: int, code_bg: int*)

classmethod get_default() → [ColorDefault](#)

Returns

Fallback instance of [Color](#) inheritor (if registries are empty).

classmethod find_closest(*hex_value: int*) → [ColorDefault](#)

Wrapper for [Approximator.find_closest\(\)](#).

Attention: Use this method only if you know what you are doing. *Default* mode colors may vary in a huge range depending on user terminal setup (colors even can have exactly the opposite value of what's listed in preset list). Much more reliable and predictable approach is to use [ColorIndexed.find_closest](#) instead.

Parameters

hex_value – Integer color value in 0xffffffff format.

Returns

Nearest found [ColorDefault](#) instance.

```
>>> ColorDefault.find_closest(0x660000)
ColorDefault[fg=31, bg=41, 0x800000]
```

to_sgr(*bg: bool = False*) → [SequenceSGR](#)

format_value(*prefix: str | None = '0x', noop_str: str = '^'*) → str

property hex_value: int | None

static hex_value_to_hsv_channels(*hex_value: int*) → Tuple[int, float, float]

Transforms **hex_value** in 0xffffffff format into tuple of three numbers corresponding to *hue*, *saturation* and *value* channel values respectively. *Hue* is within [0, 359] range, *saturation* and *value* are within [0; 1] range.

static hex_value_to_rgb_channels(*hex_value: int*) → Tuple[int, int, int]

Transforms **hex_value** in 0xffffffff format into tuple of three integers corresponding to *red*, *blue* and *green* channel value respectively. Values are within [0; 255] range.

```
>>> Color.hex_value_to_rgb_channels(0x80ff80)
(128, 255, 128)
>>> Color.hex_value_to_rgb_channels(0x000001)
(0, 0, 1)
```

class pytermor.color.ColorIndexed

Bases: [Color](#)

index: Dict[int, [ColorIndexed](#)] = {}

__init__(*hex_value: int, code: int, use_for_approximations=True*)

classmethod `get_default()` → *ColorIndexed*

Returns

Fallback instance of *Color* inheritor (if registries are empty).

classmethod `find_closest(hex_value: int)` → *ColorIndexed*

Wrapper for *Approximator.find_closest()*.

Note: Approximation algorithm ignores colors 000-015 from the *indexed* palette and will return colors with int codes in 016-255 range only. The reason for this is the same as for discouraging the usage of *ColorDefault* method version – because aforementioned colors actually depend on end-user terminal settings and the final result can be differ drastically from what's the developer imagined.

Parameters

hex_value – Integer color value in `0xffffffff` format.

Returns

Nearest found *ColorIndexed* instance.

```
>>> ColorIndexed.find_closest(0xd9dbdb)
ColorIndexed[code=253, 0xdadada]
```

to_sgr(*bg: bool = False*) → *SequenceSGR*

format_value(*prefix: str | None = '0x', noop_str: str = '^'*) → *str*

property `hex_value: int | None`

static `hex_value_to_hsv_channels(hex_value: int)` → *Tuple[int, float, float]*

Transforms `hex_value` in `0xffffffff` format into tuple of three numbers corresponding to *hue*, *saturation* and *value* channel values respectively. *Hue* is within [0, 359] range, *saturation* and *value* are within [0; 1] range.

static `hex_value_to_rgb_channels(hex_value: int)` → *Tuple[int, int, int]*

Transforms `hex_value` in `0xffffffff` format into tuple of three integers corresponding to *red*, *blue* and *green* channel value respectively. Values are within [0; 255] range.

```
>>> Color.hex_value_to_rgb_channels(0x80ff80)
(128, 255, 128)
>>> Color.hex_value_to_rgb_channels(0x000001)
(0, 0, 1)
```

class `pytermor.color.ColorRGB`

Bases: *Color*

classmethod `get_default()` → *ColorRGB*

Returns

Fallback instance of *Color* inheritor (if registries are empty).

classmethod `find_closest(hex_value: int)` → *ColorRGB*

In case of *ColorRGB* we suppose that user's terminal is not limited to a palette, therefore RGB-type color map works by simplified algorithm – it just checks if instance with same hex value was already created and returns it if that's the case, or returns a brand new instance with specified color value otherwise.

Parameters

hex_value – Integer color value in 0xffffffff format.

Returns

Existing *ColorRGB* instance or newly created one.

```
>>> existing_color1 = ColorRGB(0x660000)
>>> existing_color2 = ColorRGB(0x660000)
>>> existing_color1 == existing_color2
True
>>> existing_color1 is existing_color2 # different instances
False
>>> existing_color1 == ColorRGB.find_closest(0x660000)
True
>>> existing_color1 is ColorRGB.find_closest(0x660000) # same instance
True
```

to_sgr(bg: bool = False) → *SequenceSGR*

__init__(hex_value: Optional[int] = None, use_for_approximations: bool = True)

format_value(prefix: str | None = '0x', noop_str: str = '^') → str

property hex_value: int | None

static hex_value_to_hsv_channels(hex_value: int) → Tuple[int, float, float]

Transforms hex_value in 0xffffffff format into tuple of three numbers corresponding to *hue*, *saturation* and *value* channel values respectively. *Hue* is within [0, 359] range, *saturation* and *value* are within [0; 1] range.

static hex_value_to_rgb_channels(hex_value: int) → Tuple[int, int, int]

Transforms hex_value in 0xffffffff format into tuple of three integers corresponding to *red*, *blue* and *green* channel value respectively. Values are within [0; 255] range.

```
>>> Color.hex_value_to_rgb_channels(0x80ff80)
(128, 255, 128)
>>> Color.hex_value_to_rgb_channels(0x000001)
(0, 0, 1)
```

pytermor.color.TypeColor

Any non-abstract *Color* type.

alias of TypeVar('TypeColor', ColorDefault, ColorIndexed, ColorRGB)

class pytermor.color.Approximator

Bases: Generic[*TypeColor*]

Class contains a dictionary of registered *Colors* indexed by hex code along with cached nearest color search results to avoid unnecessary instance copies and search repeating.

__init__(parent_type: *TypeColor*)

Called in *Color*-type class constructors. Each *Color* type should have class variable with instance of *Approximator* and create it by itself if it's not present.

Parameters

parent_type – Parent *Color* type.

add_to_map(*color: TypeColor*)

Called in *Color*-type class constructors. Add a new element in color lookup table if it wasn't there, and then drop cached search results as they are most probably useless after registering a new color (i.e. now there will be better result for at least one cached value).

Parameters

color – *Color* instance being created.

get_exact(*hex_value: int*) → Optional[TypeColor]

Public interface for searching exact values in the *lookup table*, or global registry of created instances of specified *Color* class.

Parameters

hex_value – Color value in RGB format.

Returns

Color with specified value. Type is equal to type of the parent of selected color map.

find_closest(*hex_value: int*) → TypeColor

Search for nearest to *hex_value* registered color. Is used by *SGRRenderer* to find supported color alternatives in case user's terminal is incapable of operating in better mode.

Parameters

hex_value – Color value in RGB format.

Returns

Nearest to *hex_value* registered *Color*. Type is equal to type of the parent of selected color map. If no colors of required type were created (table and cache are empty), invokes *get_default()* *Color* method.

approximate(*hex_value: int, max_results: int = 1*) → List[TypeColor]

Core color approximation method. Iterate the registered SGRs table, or *lookup table*, containing parents' instances, and compute the euclidean distance from argument to each color of the palette. Sort the results and return the first <*max_results*> of them.

Note: It's not guaranteed that this method will **always** succeed in searching (the result list can be empty). Consider using *find_closest* instead, if you really want to be sure that at least some color will be returned. Another option is to use special "color" named *NOOP*.

Parameters

- **hex_value** – Color RGB value.
- **max_results** – Maximum amount of values to return.

Returns

Closest *Color* instances found, sorted by color distance descending (i.e. 0th element is always the closest to the input value).

static **__new__**(*cls, *args, **kws*)

`pytermor.color.NOOP = ColorRGB[^]`

Special instance of *Color* class always rendering into empty string.

Attention: Preset constants are omitted from API doc pages to improve readability and avoid duplication. Summary list of all presets can be found in *Preset list* section of the guide.

2.2 intcode

Module with SGR param integer codes, contains a complete or almost complete list of reliably working ones.

Suitable for *Span* and *SequenceSGR* default constructors.

Attention: Preset constants are omitted from API doc pages to improve readability and avoid duplication. Summary list of all presets can be found in *Preset list* section of the guide.

2.3 renderer

Module with output formatters. By default *SGRRenderer* is used. It also contains compatibility settings, see *SGRRenderer.set_up()*.

Working with non-default renderer can be achieved in two ways:

- a. Method *RendererManager.set_up()* sets the default renderer globally.
- b. Alternatively, you can use renderer's own class method *Renderer.render()* directly and avoid calling *Style.render()* method whatsoever.

class `pytermor.renderer.RendererManager`

classmethod `set_up(default_renderer: Type[Renderer] | None = None)`

Set up renderer preferences. Affects all renderer types.

Parameters

default_renderer – Default renderer to use globally. Passing None will result in library default setting restored (*SGRRenderer*).

```
>>> RendererManager.set_up(DebugRenderer)
>>> Style(fg='red').render('text')
'|31|text|39|'
```

```
>>> NoOpRenderer.render('text', Style(fg='red'))
'text'
```

classmethod `get_default() → Type[Renderer]`

Get global default renderer type.

class `pytermor.renderer.Renderer`

Abstract ancestor of all renderers.

abstract classmethod `render(text: Any, style: Style) → str`

Apply colors and attributes described in `style` argument to `text` and return the result. Output format depends on renderer's class (which defines the implementation).

class `pytermor.renderer.SGRRenderer`

Bases: *Renderer*

Default renderer that *Style.render()* invokes. Transforms *Color* instances defined in `style` argument into ANSI control sequence characters and merges them with input string.

classmethod `set_up`(*force_styles: bool | None = False, compatibility_indexed: bool = False, compatibility_default: bool = False*)

Set up renderer preferences. Affects all renderer types.

Parameters

- **`force_styles`** –
 - If set to *None*, all renderers will pass input text through themselves without any changes (i.e. no colors and attributes will be applied).
 - If set to *True*, renderers will always apply the formatting regardless of other internal rules and algorithms.
 - If set to *False* [default], the final decision will be made by every renderer independently, based on their own algorithms.
- **`compatibility_indexed`** – Disable *RGB* (or True Color) output mode. 256-color (*indexed*) sequences will be printed out instead of disabled ones. Useful when combined with *curses* – that way you can check the terminal capabilities from the inside of that terminal and switch to different output mode at once.
- **`compatibility_default`** – Disable *indexed* output mode and use *default* 16-color sequences instead. If this setting is set to *True*, the value of `compatibility_indexed` will be ignored completely. Useful when combined with *curses* – that way you can check the terminal capabilities from the inside of that terminal and switch to different output mode at once.

classmethod `render`(*text: Any, style: Style*)

Render text with *style* applied as ANSI control sequences.

Respects compatibility preferences (see [`RendererManager.set_up\(\)`](#)) and maps RGB colors to closest *indexed* colors if terminal doesn't support RGB output. In case terminal doesn't support even 256 colors, falls back to *default* colors, searching for closest counterparts in 16-color table.

Type of output `SequenceSGR` depends on type of *Color* variables in *style* argument. Keeping all that in mind, let's summarize:

1. *ColorRGB* can be rendered as True Color sequence, indexed sequence or default (16-color) sequence depending on terminal capabilities.
2. *ColorIndexed* can be rendered as indexed sequence or default sequence.
3. *ColorDefault* will be rendered as default-color sequence.

```
>>> SGRRenderer.render('text', Style(fg='red', bold=True))
'\x1b[1;31mtext\x1b[22;39m'
```

Parameters

- **`style`** – Style to apply.
- **`text`** – Input string.

Returns

Input string enclosed in SGR sequences.

class `pytermor.renderer.TmuxRenderer`

Bases: [`Renderer`](#)

abstract classmethod `render(text: Any, style: Style) → str`

Apply colors and attributes described in `style` argument to `text` and return the result. Output format depends on renderer's class (which defines the implementation).

class `pytermor.renderer.NoOpRenderer`

Bases: [`Renderer`](#)

classmethod `render(text: Any, style: Style) → str`

Special renderer type that does nothing with the input string and just returns it as is. That's true only when it `_is_` a `str` beforehand; otherwise argument will be casted to `str` and then returned.

```
>>> NoOpRenderer.render('text', Style(fg='red', bold=True))
'text'
```

Parameters

- **style** – Style to ignore.
- **text** – Input string.

Returns

Input string without changes.

class `pytermor.renderer.HtmlRenderer`

Bases: [`Renderer`](#)

`DEFAULT_ATTRS = ['color', 'background-color', 'font-weight', 'font-style', 'text-decoration', 'border', 'filter']`

classmethod `render(text: Any, style: Style) → str`

```
>>> HtmlRenderer.render('text', Style(fg='red', bold=True))
'<span style="color: #800000; font-weight: 700">text</span>'
```

class `pytermor.renderer.DebugRenderer`

Bases: [`Renderer`](#)

classmethod `render(text: Any, style: Style) → str`

```
>>> DebugRenderer.render('text', Style(fg='red', bold=True))
'|1;31|text|22;39|'
```

2.4 sequence

Module contains definitions for low-level ANSI escape sequences handling.

Each preset defined below is a valid argument for [`Span`](#) and [`SequenceSGR`](#) default constructors (case-insensitive).

```
>>> Span(sequence.BG_GREEN, sequence.UNDERLINED)
Span[SGR[42;4], SGR[49;24]]
```

`pytermor.sequence.build(*args: str | int | SequenceSGR) → SequenceSGR`

Create new [`SequenceSGR`](#) with specified args as params.

Resulting sequence param order is same as an argument order.

Each sequence param can be specified as:

- string key (see [span](#))
- integer param value (from [intcode](#))
- existing SequenceSGR instance (params will be extracted).

Deprecated since version 2.0: @TODO ,

```
>>> build('yellow', 'bold')
SGR[33;1]
>>> build(91, 7)
SGR[91;7]
>>> build(HI_CYAN, UNDERLINED)
SGR[96;4]
```

`pytermor.sequence.color_indexed(color: int, bg: bool = False) → SequenceSGR`

Wrapper for creation of [SequenceSGR](#) that sets foreground (or background) to one of 256-color palette value.

Deprecated since version 2.0: @TODO

Parameters

- **color** – Index of the color in the palette, 0 – 255.
- **bg** – Set to *True* to change the background color (default is foreground).

Returns

[SequenceSGR](#) with required params.

`pytermor.sequence.color_rgb(r: int, g: int, b: int, bg: bool = False) → SequenceSGR`

Wrapper for creation of [SequenceSGR](#) operating in True Color mode (16M). Valid values for *r*, *g* and *b* are in range [0; 255]. This range linearly translates into [0x00; 0xFF] for each channel. The result value is composed as #RRGGBB. For example, sequence with color of #FF3300 can be created with:

```
color_rgb(255, 51, 0)
```

Deprecated since version 2.0: @TODO

Parameters

- **r** – Red channel value, 0 – 255.
- **g** – Blue channel value, 0 – 255.
- **b** – Green channel value, 0 – 255.
- **bg** – Set to *True* to change the background color (default is foreground).

Returns

[SequenceSGR](#) with required params.

class `pytermor.sequence.Sequence`

Abstract ancestor of all escape sequences.

__init__(*params: int)

abstract encode() → str

Build up actual byte sequence and return as an ASCII-encoded string.

property params: `List[int]`

Return internal params as array.

class `pytermor.sequence.SequenceCSI`

Bases: `Sequence`

Abstract class representing CSI-type ANSI escape sequence. All subtypes of this sequence start with `e[`.

__init__(*params: `int`)

classmethod `regexp()` \rightarrow `str`

abstract `encode()` \rightarrow `str`

Build up actual byte sequence and return as an ASCII-encoded string.

property params: `List[int]`

Return internal params as array.

class `pytermor.sequence.SequenceSGR`

Bases: `SequenceCSI`

Class representing SGR-type escape sequence with varying amount of parameters.

`SequenceSGR` with zero params was specifically implemented to translate into empty string and not into `e[m`, which would have made sense, but also would be very entangling, as this sequence is equivalent of `e[0m` – hard reset sequence. The empty-string-sequence is predefined as `NOOP`.

It's possible to add of one SGR sequence to another:

```
>>> SequenceSGR(31) + SequenceSGR(1) == SequenceSGR(31, 1)
True
```

encode() \rightarrow `str`

Build up actual byte sequence and return as an ASCII-encoded string.

__init__(*params: `int`)

property params: `List[int]`

Return internal params as array.

classmethod `regexp()` \rightarrow `str`

`pytermor.sequence.NOOP = SGR[^]`

Special sequence in case where you *have to* provide one or another SGR, but do not want any control sequence to be actually included.

- `NOOP.encode()` returns empty string.
- `NOOP.params` returns empty list.

```
>>> NOOP.encode()
''
>>> NOOP.params
[]
```

New in version 1.8.

```
pytermor.sequence.RESET = SGR[0]
```

Resets all attributes and colors.

Attention: Preset constants are omitted from API doc pages to improve readability and avoid duplication. Summary list of all presets can be found in [Preset list](#) section of the guide.

2.5 span

Module introducing [Span](#) low-level abstractions. The key difference between them and Sequences is that sequence can *open* text style and also *close*, or terminate it. As for Spans – they always do both; typical use-case of [Span](#) is to wrap some text in opening SGR and closing one.

```
class pytermor.span.Span
```

```
__init__(*opening_params: str | int | SequenceSGR)
```

Create a [Span](#) with specified control sequence(s) as an opening sequence and **automatically compose** second (closing) sequence that will terminate attributes defined in the first one while keeping the others (*soft* reset).

Resulting sequence param order is same as an argument order.

Each argument can be specified as:

- string key (name of any constant defined in [intcode](#), case-insensitive)
- integer param value (defined in [intcode](#))
- existing [SequenceSGR](#) instance (params will be extracted).

```
>>> Span('red', 'bold')
Span[SGR[31;1], SGR[39;22]]
>>> Span(intcode.GREEN)
Span[SGR[32], SGR[39]]
>>> Span(93, 4)
Span[SGR[93;4], SGR[39;24]]
>>> Span(sequence.BG_BLACK + sequence.RED)
Span[SGR[40;31], SGR[49;39]]
```

Parameters

opening_params – string keys, integer codes or existing [SequenceSGR](#) instances to build [Span](#) from.

```
classmethod from_seq(opening_seq: Optional[SequenceSGR] = None, closing_seq:
    Optional[SequenceSGR] = None, hard_reset_after: bool = False) → Span
```

Create new [Span](#) with explicitly specified opening and closing sequences.

Note: [closing_seq](#) gets overwritten with [sequence.RESET](#) if [hard_reset_after](#) is *True*.

Parameters

- **opening_seq** – Starter sequence, in general determining how [Span](#) will actually look like.
- **closing_seq** – Finisher SGR sequence.

- **hard_reset_after** – Terminate *all* formatting after this span.

wrap(text: Optional[Any] = None) → str

Wrap given text string with SGRs defined on initialization – *opening_seq* on the left, *closing_seq* on the right. `str(text)` will be invoked for all argument types with the exception of *None*, which will be replaced with an empty string.

Parameters

text – String to wrap.

Returns

text enclosed in instance's SGRs, if any.

property opening_str: str

Return opening SGR sequence encoded.

property opening_seq: SequenceSGR

Return opening SGR sequence instance.

property closing_str: str

Return closing SGR sequence encoded.

property closing_seq: SequenceSGR

Return closing SGR sequence instance.

__call__(text: Optional[Any] = None) → str

Can be used instead of *wrap()* method.

```
>>> RED('text') == RED.wrap('text')
True
```

`pytermor.span.NOOP = Span[SGR[^], SGR[^]]`

Special *Span* in cases where you *have to* select one or another *Span*, but do not want any control sequence to be actually included.

- `NOOP(string)` or `NOOP.wrap(string)` returns string without any modifications;
- `NOOP.opening_str` and `NOOP.closing_str` are empty strings;
- `NOOP.opening_seq` and `NOOP.closing_seq` both returns *sequence.NOOP*.

```
>>> NOOP('text')
'text'
>>> NOOP.opening_str
''
>>> NOOP.opening_seq
SGR[^]
```

Attention: Preset constants are omitted from API doc pages to improve readability and avoid duplication. Summary list of all presets can be found in *Preset list* section of the guide.

2.6 style

High-level abstraction defining text colors and attributes.

`class pytermor.style.Style`

Create a new `Style()`.

Key difference between `Styles` and `Spans` or `SGRs` is that `Styles` describe colors in RGB format and therefore support output rendering in several different formats (see [renderer](#)).

Both `fg` and `bg` can be specified as:

1. `Color` instance or library preset;
2. key code – name of any of aforementioned presets, case-insensitive;
3. integer color value in hexademical RGB format.
4. `None` – the color will be unset.

Parameters

- **fg** – Foreground (i.e., text) color.
- **bg** – Background color.
- **inherit** – Parent instance to copy unset properties from.
- **blink** – Blinking effect; *supported by limited amount of Renderers*.
- **bold** – Bold or increased intensity.
- **crosslined** – Strikethrough.
- **dim** – Faint, decreased intensity.
- **double_underlined** – Faint, decreased intensity.
- **inversed** – Swap foreground and background colors.
- **italic** – Italic.
- **overlined** – Overline.
- **underlined** – Underline.
- **class_name** – Arbitrary string used by some renderers, e.g. by `HtmlRenderer`.

```
>>> Style(fg='green', bold=True)
Style[fg=008000, no bg, bold]
>>> Style(bg=0x0000ff)
Style[no fg, bg=0000ff]
>>> Style(fg=color.XTERM_DEEP_SKY_BLUE_1, bg=color.XTERM_GREY_93)
Style[fg=00afff, bg=eeeeee]
```

```
__init__(inherit: Style = None, fg: Color | int | str = None, bg: Color | int | str = None, blink: bool = None,
         bold: bool = None, crosslined: bool = None, dim: bool = None, double_underlined: bool = None,
         inversed: bool = None, italic: bool = None, overlined: bool = None, underlined: bool = None,
         class_name: str = None)
```

render(*text: Optional[Any] = None*) → str

Returns *text* with all attributes and colors applied.

By default uses [SequenceSGR](#) renderer, that means that output will contain ANSI escape sequences.

autopick_fg() → [Color](#) | None

Pick *fg_color* depending on *bg_color* brightness. Set 4% gray or 96% gray as *fg_color* and return it, if *bg_color* is defined, and do nothing and return None otherwise.

Returns

Suitable foreground color or None.

property fg: [Color](#)

property bg: [Color](#)

class pytermor.style.Stylesheet

whatwhenhow @FIXME

__init__(*default: Optional[Style] = None*)

2.7 text

class pytermor.text.Text

__init__(*text: Any = None, style: Style | str = None*)

render() → str

append(*text: str | Text*)

prepend(*text: str | Text*)

pytermor.text.distribute_padded(*values: List[str | Text], max_len: int, pad_before: bool = False, pad_after: bool = False*) → str

2.8 util

Package containing a set of formatters for prettier output, as well as utility classes for removing some of the boilerplate code when dealing with escape sequences.

pytermor.util.format_thousand_sep(*value: int | float, separator=' '*)

Returns input value with integer part splitted into groups of three digits, joined then with *separator* string.

```
>>> format_thousand_sep(260341)
'260 341'
>>> format_thousand_sep(-9123123123.55, ',')
'-9,123,123,123.55'
```

2.8.1 auto_float

`pytermor.util.auto_float.format_auto_float(value: float, req_len: int, allow_exponent_notation: bool = True) → str`

Dynamically adjust decimal digit amount and format to fill up the output string with as many significant digits as possible, and keep the output length strictly equal to `req_len` at the same time.

```
>>> format_auto_float(0.016789, 5)
'0.017'
>>> format_auto_float(0.167891, 5)
'0.168'
>>> format_auto_float(1.567891, 5)
'1.568'
>>> format_auto_float(12.56789, 5)
'12.57'
>>> format_auto_float(123.5678, 5)
'123.6'
>>> format_auto_float(1234.567, 5)
' 1235'
>>> format_auto_float(12345.67, 5)
'12346'
```

For cases when it's impossible to fit a number in the required length and rounding doesn't help (e.g. 12 500 000 and 5 chars) algorithm switches to scientific notation and the result looks like '1.2e7'.

When exponent form is disabled, there are two options for value that cannot fit into required length:

- 1) if absolute value is less than 1, zeros will be displayed ('0.0000');
- 2) in case of big numbers (like 10^9) `ValueError` will be raised instead.

Parameters

- **value** – Value to format
- **req_len** – Required output string length
- **allow_exponent_notation** – Enable/disable exponent form.

Returns

Formatted string of required length

Raises

ValueError –

New in version 1.7.

2.8.2 prefixed_unit

`pytermor.util.prefixed_unit.format_si_metric(value: float, unit: str = 'm') → str`

Format value as meters with SI-prefixes, max result length is 6 chars. Base is 1000. Unit can be customized.

```
>>> format_si_metric(123.456)
'123 m'
>>> format_si_metric(0.331, 'g')
'331 mg'
```

(continues on next page)

(continued from previous page)

```
>>> format_si_metric(45200, 'V')
'45.2 kV'
>>> format_si_metric(1.26e-9, 'm²')
'1.26 nm²'
```

Parameters

- **value** – Input value (unitless).
- **unit** – Value unit, printed right after the prefix.

Returns

Formatted string with SI-prefix if necessary.

New in version 2.0.

`pytermor.util.prefixed_unit.format_si_binary(value: float, unit: str = 'b') → str`

Format value as binary size (bytes, kbytes, Mbytes), max result length is 8 chars. Base is 1024. Unit can be customized.

```
>>> format_si_binary(631)
'631 b'
>>> format_si_binary(1080)
'1.055 kb'
>>> format_si_binary(45200)
'44.14 kb'
>>> format_si_binary(1.258 * pow(10, 6), 'bps')
'1.200 Mbps'
```

Parameters

- **value** – Input value in bytes.
- **unit** – Value unit, printed right after the prefix.

Returns

Formatted string with SI-prefix if necessary.

New in version 2.0.

class `pytermor.util.prefixed_unit.PrefixedUnitFormatter`

Formats value using settings passed to constructor. The main idea of this class is to fit into specified string length as much significant digits as it's theoretically possible by using multipliers and unit prefixes to indicate them.

You can create your own formatters if you need fine tuning of the output and customization. If that's not the case, there are facade methods `format_si_metric()` and `format_si_binary()`, which will invoke predefined formatters and doesn't require setting up.

@TODO params desc

Parameters

prefix_zero_idx – Index of prefix which will be used as default, i.e. without multiplying coefficients.

New in version 1.7.


```
__init__(max_value_len: int, truncate_frac: bool = False, unit: str = None, unit_separator: str = None,
         mcoef: float = 1000.0, prefixes: List[str | None] = None, prefix_zero_idx: int = None)
```

property max_len: int

Returns

Maximum length of the result. Note that constructor argument is *max_value_len*, which is different parameter.

```
format(value: float, unit: Optional[str] = None) → str
```

Parameters

- **value** – Input value
- **unit** – Unit override

Returns

Formatted value

```
pytermor.util.prefixed_unit.PREFIXES_SI = ['y', 'z', 'a', 'f', 'p', 'n', '', 'm', None,
      'k', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y']
```

Prefix presets used by default module formatters. Can be useful if you are building your own formatter.

```
pytermor.util.prefixed_unit.PREFIX_ZERO_SI = 8
```

Index of prefix which will be used as default, i.e. without multiplying coefficients.

2.8.3 time_delta

Module for time difference formatting (e.g. “4 days 15 hours”, “8h 59m”).

Supports several output lengths and can be customized even more.

```
pytermor.util.time_delta.format_time_delta(seconds: float, max_len: Optional[int] = None) → str
```

Format time delta using suitable format (which depends on *max_len* argument). Key feature of this formatter is ability to combine two units and display them simultaneously, e.g. return “3h 48min” instead of “228 mins” or “3 hours”,

There are predefined formatters with output length of 3, 4, 6 and 10 characters. Therefore, you can pass in any value from 3 inclusive and it’s guaranteed that result’s length will be less or equal to required length. If *max_len* is omitted, longest registered formatter will be used.

```
>>> format_time_delta(10, 3)
'10s'
>>> format_time_delta(10, 6)
'10 sec'
>>> format_time_delta(15350, 4)
'4 h'
>>> format_time_delta(15350)
'4h 15min'
```

Parameters

- **seconds** – Value to format
- **max_len** – Maximum output string length (total)

Returns

Formatted string

class pytermor.util.time_delta.TimeDeltaFormatter

Formatter for time intervals. Key feature of this formatter is ability to combine two units and display them simultaneously, e.g. return “3h 48min” instead of “228 mins” or “3 hours”, etc.

You can create your own formatters if you need fine tuning of the output and customization. If that’s not the case, there is a facade method `format_time_delta()` which will select appropriate formatter automatically.

Example output:

`"10 secs", "5 mins", "4h 15min", "5d 22h"`

__init__(*units: List[TimeUnit], allow_negative: bool, unit_separator: Optional[str] = None, plural_suffix: Optional[str] = None, stylesheet: Optional[TimeDeltaStylesheet] = None, overflow_msg: str = 'OVERFLOW'*)

property stylesheet: *TimeDeltaStylesheet*

property max_len: *int*

This property cannot be set manually, it is computed on initialization automatically.

Returns

Maximum possible output string length.

format(*seconds: float, always_max_len: bool = False*) → *str*

Format the requested amount of seconds and apply styles to the result as defined in current formatter’s *stylesheet*. Default *stylesheet* contains no-op spans only and thus no styles will be applied.

Parameters

- **seconds** – Input value.
- **always_max_len** – If result string is less than *max_len* it will be returned as is, unless this flag is set to *True*. In that case output string will be padded with spaces on the left side so that resulting length would be always equal to maximum length.

Returns

Formatted string with applied styles if they are defined, raw string otherwise.

format_raw(*seconds: float*) → *str | None*

Format the requested amount of seconds as raw string without styling.

Parameters

seconds – Input value.

Returns

Formatted string or *None* on overflow (if input value is too big for the current formatter to handle).

class pytermor.util.time_delta.TimeUnit

TimeUnit(name: ‘str’, in_next: ‘int’ = None, custom_short: ‘str’ = None, collapsible_after: ‘int’ = None, overflow_afer: ‘int’ = None)

name: *str*

in_next: *int* = None

custom_short: *str* = None

collapsible_after: *int* = None

```
overflow_afer: int = None
```

```
__init__(name: str, in_next: Optional[int] = None, custom_short: Optional[str] = None, collapsible_after:
Optional[int] = None, overflow_afer: Optional[int] = None) → None
```

```
class pytermor.util.time_delta.TimeDeltaStylesheet
```

```
Bases: Stylesheet
```

```
__init__(default: Optional[Style] = None, digit: Optional[Style] = None, unit: Optional[Style] = None,
overflow: Optional[Style] = None)
```

2.8.4 stdlib_ext

Some of the Python Standard Library methods rewritten for correct work with strings containing control sequences.

```
pytermor.util.stdlib_ext.ljust_sgr(s: str, width: int, fillchar: str = ' ') → str
```

SGR-formatting-aware implementation of `str.ljust`.

Return a left-justified string of length `width`. Padding is done using the specified fill character (default is a space).

```
pytermor.util.stdlib_ext.rjust_sgr(s: str, width: int, fillchar: str = ' ') → str
```

SGR-formatting-aware implementation of `str.rjust`.

Return a right-justified string of length `width`. Padding is done using the specified fill character (default is a space).

```
pytermor.util.stdlib_ext.center_sgr(s: str, width: int, fillchar: str = ' ') → str
```

SGR-formatting-aware implementation of `str.center`.

Return a centered string of length `width`. Padding is done using the specified fill character (default is a space).

```
@ TODO    (.)# - f-
```

2.8.5 string_filter

String filtering module.

Main idea is to provide a common interface for string filtering, that can make possible working with filters like with objects rather than with functions/lambda's.

```
pytermor.util.string_filter.apply_filters(s: AnyStr, *args: StringFilter[AnyStr] |
Type[StringFilter[AnyStr]]) → AnyStr
```

Method for applying dynamic filter list to a target string/bytes. Example (will replace all ESC control characters to E and thus make SGR params visible):

```
>>> apply_filters(span.RED('test'), ReplaceSGR(r'E\2\3\5'))
'E[31mtestE[39m'
```

Note that type of `s` argument must be same as `StringFilter` parameterized type, i.e. `ReplaceNonAsciiBytes` is `StringFilter[bytes]` type, so you can apply it only to bytes-type strings.

Parameters

- `s` (`AnyStr`) – String to filter.
- `args` – `StringFilter` instance(s) or `StringFilter` class(es).

Returns

Filtered s.

class pytermor.util.string_filter.**StringFilter**Bases: `Generic`

Common string modifier interface.

__init__(*pattern: AnyStr, repl: AnyStr | Callable[[AnyStr | Match], AnyStr]*)**__call__**(*s: AnyStr*) → `AnyStr`Can be used instead of *apply()***apply**(*s: AnyStr*) → `AnyStr`

Apply filter to s string (or bytes).

static **__new__**(*cls, *args, **kws*)**class** pytermor.util.string_filter.**VisualuzeWhitespace**Bases: `StringFilter[str]`Replace every invisible character with *repl* (default is `.`), except newlines. Newlines are kept and get prepended with same string.

```
>>> VisualuzeWhitespace().apply('A B C')
'A..B..C'
>>> apply_filters('1. D\n2. L ', VisualuzeWhitespace)
'1..D.\n2..L.'
```

__init__(*repl: AnyStr = '.'*)**__call__**(*s: AnyStr*) → `AnyStr`Can be used instead of *apply()***static** **__new__**(*cls, *args, **kws*)**apply**(*s: AnyStr*) → `AnyStr`

Apply filter to s string (or bytes).

class pytermor.util.string_filter.**ReplaceSGR**Bases: `StringFilter[str]`Find all SGR seqs (e.g. `ESC[1;4m`) and replace with given string. More specific version of [ReplaceCSI](#).**Parameters****repl** – Replacement, can contain regexp groups (see [apply_filters\(\)](#)).**__init__**(*repl: AnyStr = ''*)**__call__**(*s: AnyStr*) → `AnyStr`Can be used instead of *apply()***static** **__new__**(*cls, *args, **kws*)**apply**(*s: AnyStr*) → `AnyStr`

Apply filter to s string (or bytes).

class pytermor.util.string_filter.ReplaceCSIBases: [StringFilter](#)[str]

Find all CSI seqs (i.e. ESC[*) and replace with given string. Less specific version of [ReplaceSGR](#), as CSI consists of SGR and many other sequence subtypes.

Parameters

repl – Replacement, can contain regexp groups (see [apply_filters\(\)](#)).

__init__(*repl: AnyStr* = "")

__call__(*s: AnyStr*) → AnyStr

Can be used instead of *apply()*

static **__new__**(*cls, *args, **kws*)

apply(*s: AnyStr*) → AnyStr

Apply filter to s string (or bytes).

class pytermor.util.string_filter.ReplaceNonAsciiBytesBases: [StringFilter](#)[bytes]

Keep 7-bit ASCII bytes [0x00 - 0x7f], replace other to ?.

Parameters

repl – Replacement byte-string.

__init__(*repl: AnyStr* = b'?')

__call__(*s: AnyStr*) → AnyStr

Can be used instead of *apply()*

static **__new__**(*cls, *args, **kws*)

apply(*s: AnyStr*) → AnyStr

Apply filter to s string (or bytes).

CHANGELOG

3.1 v2.0.0

- Complete library rewrite.
- High-level abstractions *Color*, *Renderer* and *Style*.
- Unit tests for formatters and new modules.
- pytest and coverage integration.
- sphinx and readthedocs integraton.

3.2 v1.8.0

- `format_prefixed_unit` extended for working with decimal and binary metric prefixes.
- `format_time_delta` extended with new settings.
- Value rounding transferred from `format_auto_float` to `format_prefixed_unit`.
- Utility classes reorganization.
- Unit tests output formatting.
- `sequence.NOOP` SGR sequence and `span.NOOP` format.
- Max decimal points for `auto_float` extended from (2) to (max-2).

3.3 v1.7.4

- Added 3 formatters: `format_prefixed_unit`, `format_time_delta`, `format_auto_float`.

3.4 v1.7.3

- Added `span.BG_BLACK` format.

3.5 v1.7.2

- Added `ljust_sgr`, `rjust_sgr`, `center_sgr` util functions to align strings with SGRs correctly.

3.6 v1.7.1

- Print reset sequence as `\e[m` instead of `\e[0m`.

3.7 v1.7.0

- `Span` constructor can be called without arguments.
- Added SGR code lists.

3.8 v1.6.2

- Excluded `tests` dir from distribution package.

3.9 v1.6.1

- Ridded of `EmptyFormat` and `AbstractFormat` classes.
- Renamed code module to `sgr` because of conflicts in PyCharm debugger (`pydevd_console_integration.py`).

3.10 v1.5.0

- Removed excessive `EmptySequenceSGR` – default SGR class was specifically implemented to print out as empty string instead of `\e[m` if constructed without params.

3.11 v1.4.0

- `Span.wrap()` now accepts any type of argument, not only `str`.
- Rebuilt Sequence inheritance tree.
- Added equality methods for `SequenceSGR` and `Span` classes/subclasses.
- Added some tests for `fmt.*` and `seq.*` classes.

3.12 v1.3.2

- Added `span.GRAY` and `span.BG_GRAY` format presets.

3.13 v1.3.1

- Interface revisioning.

3.14 v1.2.1

- `opening_seq` and `closing_seq` properties for `Span` class.

3.15 v1.2.0

- `EmptySequenceSGR` and `EmptyFormat` classes.

3.16 v1.1.0

- Autoformat feature.

3.17 v1.0.0

- First public version.

This project uses Semantic Versioning – <https://semver.org> (*starting from 2.0.0*)

LICENSE**MIT License**

Copyright (c) 2022 Aleksandr Shavykin

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PYTHON MODULE INDEX

p

- `pytermor.color`, 21
- `pytermor.intcode`, 26
- `pytermor.renderer`, 26
- `pytermor.sequence`, 28
- `pytermor.span`, 31
- `pytermor.style`, 33
- `pytermor.text`, 34
- `pytermor.util`, 34
 - `pytermor.util.auto_float`, 35
 - `pytermor.util.prefixed_unit`, 35
 - `pytermor.util.stdlib_ext`, 39
 - `pytermor.util.string_filter`, 39
 - `pytermor.util.time_delta`, 37

Symbols

- `__call__()` (*pytermor.span.Span* method), 32
 - `__call__()` (*pytermor.util.string_filter.ReplaceCSI* method), 41
 - `__call__()` (*pytermor.util.string_filter.ReplaceNonAsciiBytes* method), 41
 - `__call__()` (*pytermor.util.string_filter.ReplaceSGR* method), 40
 - `__call__()` (*pytermor.util.string_filter.StringFilter* method), 40
 - `__call__()` (*pytermor.util.string_filter.VisualuzeWhitespace* method), 40
 - `__init__()` (*pytermor.color.Approximator* method), 24
 - `__init__()` (*pytermor.color.Color* method), 21
 - `__init__()` (*pytermor.color.ColorDefault* method), 22
 - `__init__()` (*pytermor.color.ColorIndexed* method), 22
 - `__init__()` (*pytermor.color.ColorRGB* method), 24
 - `__init__()` (*pytermor.sequence.Sequence* method), 29
 - `__init__()` (*pytermor.sequence.SequenceCSI* method), 30
 - `__init__()` (*pytermor.sequence.SequenceSGR* method), 30
 - `__init__()` (*pytermor.span.Span* method), 31
 - `__init__()` (*pytermor.style.Style* method), 33
 - `__init__()` (*pytermor.style.Stylesheet* method), 34
 - `__init__()` (*pytermor.text.Text* method), 34
 - `__init__()` (*pytermor.util.prefixed_unit.PrefixedUnitFormatter* method), 36
 - `__init__()` (*pytermor.util.string_filter.ReplaceCSI* method), 41
 - `__init__()` (*pytermor.util.string_filter.ReplaceNonAsciiBytes* method), 41
 - `__init__()` (*pytermor.util.string_filter.ReplaceSGR* method), 40
 - `__init__()` (*pytermor.util.string_filter.StringFilter* method), 40
 - `__init__()` (*pytermor.util.string_filter.VisualuzeWhitespace* method), 40
 - `__init__()` (*pytermor.util.time_delta.TimeDeltaFormatter* method), 38
 - `__init__()` (*pytermor.util.time_delta.TimeDeltaStylesheet* method), 39
 - `__init__()` (*pytermor.util.time_delta.TimeUnit* method), 39
 - `__new__()` (*pytermor.color.Approximator* static method), 25
 - `__new__()` (*pytermor.util.string_filter.ReplaceCSI* static method), 41
 - `__new__()` (*pytermor.util.string_filter.ReplaceNonAsciiBytes* static method), 41
 - `__new__()` (*pytermor.util.string_filter.ReplaceSGR* static method), 40
 - `__new__()` (*pytermor.util.string_filter.StringFilter* static method), 40
 - `__new__()` (*pytermor.util.string_filter.VisualuzeWhitespace* static method), 40
- ## A
- `add_to_map()` (*pytermor.color.Approximator* method), 24
 - `append()` (*pytermor.text.Text* method), 34
 - `apply()` (*pytermor.util.string_filter.ReplaceCSI* method), 41
 - `apply()` (*pytermor.util.string_filter.ReplaceNonAsciiBytes* method), 41
 - `apply()` (*pytermor.util.string_filter.ReplaceSGR* method), 40
 - `apply()` (*pytermor.util.string_filter.StringFilter* method), 40
 - `apply()` (*pytermor.util.string_filter.VisualuzeWhitespace* method), 40
 - `apply_filters()` (in module *pytermor.util.string_filter*), 39
 - `approximate()` (*pytermor.color.Approximator* method), 25
 - `Approximator` (class in *pytermor.color*), 24
 - `autopick_fg()` (*pytermor.style.Style* method), 34
- ## B
- `bg` (*pytermor.style.Style* property), 34
 - `build()` (in module *pytermor.sequence*), 28
- ## C
- `center_sgr()` (in module *pytermor.util.stdlib_ext*), 39

closing_seq (*pytermor.span.Span* property), 32
 closing_str (*pytermor.span.Span* property), 32
 collapsible_after (*pytermor.util.time_delta.TimeUnit* attribute), 38
 Color (class in *pytermor.color*), 21
 color_indexed() (in module *pytermor.sequence*), 29
 color_rgb() (in module *pytermor.sequence*), 29
 ColorDefault (class in *pytermor.color*), 22
 ColorIndexed (class in *pytermor.color*), 22
 ColorRGB (class in *pytermor.color*), 23
 custom_short (*pytermor.util.time_delta.TimeUnit* attribute), 38

D

DebugRenderer (class in *pytermor.renderer*), 28
 DEFAULT_ATTRS (*pytermor.renderer.HtmlRenderer* attribute), 28
 distribute_padded() (in module *pytermor.text*), 34

E

encode() (*pytermor.sequence.Sequence* method), 29
 encode() (*pytermor.sequence.SequenceCSI* method), 30
 encode() (*pytermor.sequence.SequenceSGR* method), 30

F

fg (*pytermor.style.Style* property), 34
 find_closest() (*pytermor.color.Approximator* method), 25
 find_closest() (*pytermor.color.Color* class method), 21
 find_closest() (*pytermor.color.ColorDefault* class method), 22
 find_closest() (*pytermor.color.ColorIndexed* class method), 23
 find_closest() (*pytermor.color.ColorRGB* class method), 23
 format() (*pytermor.util.prefixed_unit.PrefixedUnitFormatter* method), 37
 format() (*pytermor.util.time_delta.TimeDeltaFormatter* method), 38
 format_auto_float() (in module *pytermor.util.auto_float*), 35
 format_raw() (*pytermor.util.time_delta.TimeDeltaFormatter* method), 38
 format_si_binary() (in module *pytermor.util.prefixed_unit*), 36
 format_si_metric() (in module *pytermor.util.prefixed_unit*), 35
 format_thousand_sep() (in module *pytermor.util*), 34
 format_time_delta() (in module *pytermor.util.time_delta*), 37
 format_value() (*pytermor.color.Color* method), 21
 format_value() (*pytermor.color.ColorDefault* method), 22

format_value() (*pytermor.color.ColorIndexed* method), 23
 format_value() (*pytermor.color.ColorRGB* method), 24
 from_seq() (*pytermor.span.Span* class method), 31

G

get_default() (*pytermor.color.Color* class method), 21
 get_default() (*pytermor.color.ColorDefault* class method), 22
 get_default() (*pytermor.color.ColorIndexed* class method), 22
 get_default() (*pytermor.color.ColorRGB* class method), 23
 get_default() (*pytermor.renderer.RendererManager* class method), 26
 get_exact() (*pytermor.color.Approximator* method), 25

H

hex_value (*pytermor.color.Color* property), 21
 hex_value (*pytermor.color.ColorDefault* property), 22
 hex_value (*pytermor.color.ColorIndexed* property), 23
 hex_value (*pytermor.color.ColorRGB* property), 24
 hex_value_to_hsv_channels() (*pytermor.color.Color* static method), 21
 hex_value_to_hsv_channels() (*pytermor.color.ColorDefault* static method), 22
 hex_value_to_hsv_channels() (*pytermor.color.ColorIndexed* static method), 23
 hex_value_to_hsv_channels() (*pytermor.color.ColorRGB* static method), 24
 hex_value_to_rgb_channels() (*pytermor.color.Color* static method), 21
 hex_value_to_rgb_channels() (*pytermor.color.ColorDefault* static method), 22
 hex_value_to_rgb_channels() (*pytermor.color.ColorIndexed* static method), 23
 hex_value_to_rgb_channels() (*pytermor.color.ColorRGB* static method), 24
 HtmlRenderer (class in *pytermor.renderer*), 28

in_next (*pytermor.util.time_delta.TimeUnit* attribute), 38
 index (*pytermor.color.ColorIndexed* attribute), 22

L

ljust_sgr() (in module *pytermor.util.stdlib_ext*), 39

M

max_len (*pytermor.util.prefixed_unit.PrefixedUnitFormatter* property), 37

`max_len` (*pytermor.util.time_delta.TimeDeltaFormatter* property), 38

module

- `pytermor.color`, 21
- `pytermor.intcode`, 26
- `pytermor.renderer`, 26
- `pytermor.sequence`, 28
- `pytermor.span`, 31
- `pytermor.style`, 33
- `pytermor.text`, 34
- `pytermor.util`, 34
- `pytermor.util.auto_float`, 35
- `pytermor.util.prefixed_unit`, 35
- `pytermor.util.stdlib_ext`, 39
- `pytermor.util.string_filter`, 39
- `pytermor.util.time_delta`, 37

N

`name` (*pytermor.util.time_delta.TimeUnit* attribute), 38

`NOOP` (in module *pytermor.color*), 25

`NOOP` (in module *pytermor.sequence*), 30

`NOOP` (in module *pytermor.span*), 32

`NoOpRenderer` (class in *pytermor.renderer*), 28

O

`opening_seq` (*pytermor.span.Span* property), 32

`opening_str` (*pytermor.span.Span* property), 32

`overflow_afer` (*pytermor.util.time_delta.TimeUnit* attribute), 38

P

`params` (*pytermor.sequence.Sequence* property), 29

`params` (*pytermor.sequence.SequenceCSI* property), 30

`params` (*pytermor.sequence.SequenceSGR* property), 30

`PREFIX_ZERO_SI` (in module *pytermor.util.prefixed_unit*), 37

`PrefixedUnitFormatter` (class in *pytermor.util.prefixed_unit*), 36

`PREFIXES_SI` (in module *pytermor.util.prefixed_unit*), 37

`prepend()` (*pytermor.text.Text* method), 34

`pytermor.color`
module, 21

`pytermor.intcode`
module, 26

`pytermor.renderer`
module, 26

`pytermor.sequence`
module, 28

`pytermor.span`
module, 31

`pytermor.style`
module, 33

`pytermor.text`
module, 34

`pytermor.util`
module, 34

`pytermor.util.auto_float`
module, 35

`pytermor.util.prefixed_unit`
module, 35

`pytermor.util.stdlib_ext`
module, 39

`pytermor.util.string_filter`
module, 39

`pytermor.util.time_delta`
module, 37

R

`regex()` (*pytermor.sequence.SequenceCSI* class method), 30

`regex()` (*pytermor.sequence.SequenceSGR* class method), 30

`render()` (*pytermor.renderer.DebugRenderer* class method), 28

`render()` (*pytermor.renderer.HtmlRenderer* class method), 28

`render()` (*pytermor.renderer.NoOpRenderer* class method), 28

`render()` (*pytermor.renderer.Renderer* class method), 26

`render()` (*pytermor.renderer.SGRRenderer* class method), 27

`render()` (*pytermor.renderer.TmuxRenderer* class method), 27

`render()` (*pytermor.style.Style* method), 33

`render()` (*pytermor.text.Text* method), 34

`Renderer` (class in *pytermor.renderer*), 26

`RendererManager` (class in *pytermor.renderer*), 26

`ReplaceCSI` (class in *pytermor.util.string_filter*), 40

`ReplaceNonAsciiBytes` (class in *pytermor.util.string_filter*), 41

`ReplaceSGR` (class in *pytermor.util.string_filter*), 40

`RESET` (in module *pytermor.sequence*), 30

`rjust_sgr()` (in module *pytermor.util.stdlib_ext*), 39

S

`Sequence` (class in *pytermor.sequence*), 29

`SequenceCSI` (class in *pytermor.sequence*), 30

`SequenceSGR` (class in *pytermor.sequence*), 30

`set_up()` (*pytermor.renderer.RendererManager* class method), 26

`set_up()` (*pytermor.renderer.SGRRenderer* class method), 26

`SGRRenderer` (class in *pytermor.renderer*), 26

`Span` (class in *pytermor.span*), 31

`StringFilter` (class in *pytermor.util.string_filter*), 40

`Style` (class in *pytermor.style*), 33

`Stylesheet` (class in *pytermor.style*), 34

stylesheet (*pytermor.util.time_delta.TimeDeltaFormatter* property), 38

T

Text (*class in pytermor.text*), 34

TimeDeltaFormatter (*class in pytermor.util.time_delta*), 38

TimeDeltaStylesheet (*class in pytermor.util.time_delta*), 39

TimeUnit (*class in pytermor.util.time_delta*), 38

TmuxRenderer (*class in pytermor.renderer*), 27

to_sgr() (*pytermor.color.Color method*), 21

to_sgr() (*pytermor.color.ColorDefault method*), 22

to_sgr() (*pytermor.color.ColorIndexed method*), 23

to_sgr() (*pytermor.color.ColorRGB method*), 24

TypeColor (*in module pytermor.color*), 24

V

VisualuzeWhitespace (*class in pytermor.util.string_filter*), 40

W

wrap() (*pytermor.span.Span method*), 32