



pytermor

Release 2.25.0-dev1

Alexandr Shavykin

Dec 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	4
1.2	High-level abstractions	6
1.2.1	ColorIndex and Styles	6
1.2.2	Output format control	6
1.2.3	Color mode fallbacks	6
1.2.4	Core API	6
1.3	Low-level abstractions	7
1.3.1	Format soft reset	7
1.3.2	Working with Spans	8
1.3.3	Creating and applying SGRs	9
1.3.4	SGR sequence structure	9
1.3.5	Combining SGRs	10
1.3.6	Core API	10
1.4	Preset list	10
1.4.1	Meta, attributes, resetters	11
1.4.2	Color16 presets	12
1.4.3	Color256 presets	13
1.5	Xterm color palette	18
1.6	Named Colors collection	20
1.6.1	lisc	20
1.7	String (and bytes) filters	20
1.7.1	filters	20
1.8	Number formatters	20
1.8.1	Auto-float formatter	20
1.8.2	Prefixed-unit formatter	20
1.8.3	Time delta formatter	20
1.9	Documentation guidelines	21
2	API reference	23
2.1	ansi	23
2.2	color	31
2.3	common	42
2.4	cval	43
2.5	renderer	51
2.6	style	58
2.7	text	59

2.8	utilmisc	60
2.9	utilnum	62
2.10	utilstr	67
3	Changelog	73
4	License	79
	Python Module Index	81

(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[3m` 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.

Diagram illustrating the components of an ANSI SGR command sequence for rendering the word "text" in red with a single underline:

- `\x1b`: ESC character in Hex ASCII
- `[`: CSI (Control Sequence Initiator)
- `31;4m`: Parameters (31 for red foreground, 4 for single underline)
- `t`: Final Byte

Notes

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

A L	Module	Class(es)	Purpose
Hi	<i>text</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> <i>Styles</i>	Reusable abstractions defining colors and text attributes (text color, bg color, <i>bold</i> attribute, <i>underlined</i> attribute etc).
		<i>SgrRenderer</i> <i>HtmlRenderer</i> <i>TmuxRenderer</i> etc.	<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 what output format is required.
	<i>color</i>	<i>Color16</i> <i>Color256</i> <i>ColorRGB</i>	Abstractions for color operations in different color modes (default 16-color, 256-color, RGB). Tools for color approximation and transformations.
		<i>pytermor</i>	Color registry.
Lo	<i>ansi</i>	<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>Spans</i>	Registry of predefined instances in case the developer doesn’t need dynamic output formatting and just wants to colorize an error message.
		<i>SequenceSGR</i> <i>SeqIndex</i>	Abstractions for manipulating ANSI control sequences and classes-factories, plus a registry of preset SGRs.
		<i>IntCodes</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 Spans
2
3 print(Spans.RED('Feat') + Spans.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* using default constructor, 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 SeqIndex, SequenceSGR
2
3 seq1 = SequenceSGR('hi_blue', 1) # keys or integer codes
4 seq2 = SequenceSGR(seq1, SeqIndex.ITALIC) # existing SGRs
5 seq3 = SequenceSGR('underlined', 'YELLOW') # case-insensitive
6
7 msg = f'{seq1}Flexible{SeqIndex.RESET} ' + \
8       f'{seq2}sequence{SeqIndex.RESET} ' + \
9       str(seq3) + 'builder' + str(SeqIndex.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 SequenceSGR, SeqIndex
2
3 start_color = 41
4 for idx, c in enumerate(range(start_color, start_color+(36*6), 36)):
5     print(f'{SequenceSGR.new_color_256(c)}{SeqIndex.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'{SequenceSGR.new_color_rgb(r, g, b)}{SeqIndex.COLOR_OFF}', end='')

```



Customizable output formats

Todo: @TODOTODO

String and number formatters

Todo: @TODOTODO

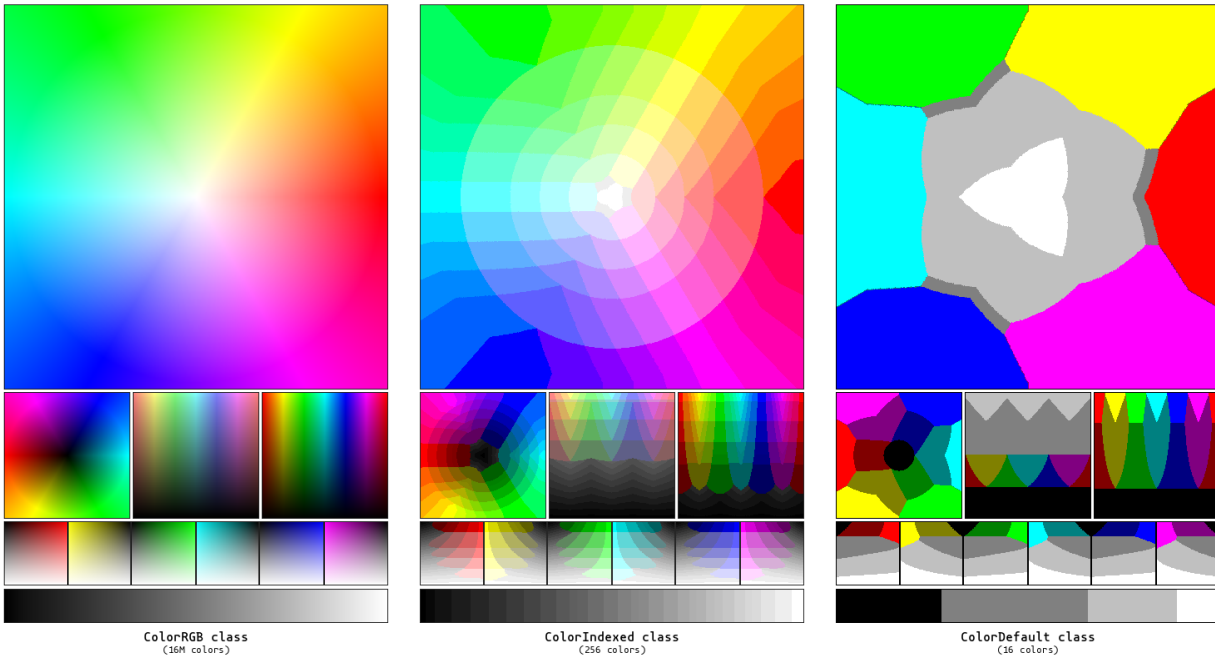


Fig. 1: Color approximations for indexed modes

1.2 High-level abstractions

1.2.1 ColorIndex and Styles

1.2.2 Output format control

1.2.3 Color mode fallbacks

1.2.4 Core API

@EXAMPLES

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 ESC [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 ESC [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 Span, Spans, SeqIndex
2
3 # implicitly:
```

(continues on next page)

(continued from previous page)

```

4 span_warn = Span(93, 4)
5 # or explicitly:
6 span_warn = Span.init_explicit(
7     SeqIndex.HI_YELLOW + SeqIndex.UNDERLINED, # sequences can be summed up, remember?
8     SeqIndex.COLOR_OFF + SeqIndex.UNDERLINED_OFF, # "counteractive" sequences
9     hard_reset_after=False
10 )
11
12 orig_text = Spans.BOLD(f'this is {SeqIndex.BG_GRAY}the original{SeqIndex.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 `SeqIndex.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.init_explicit()`.

1.3.3 Creating and applying SGRs

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

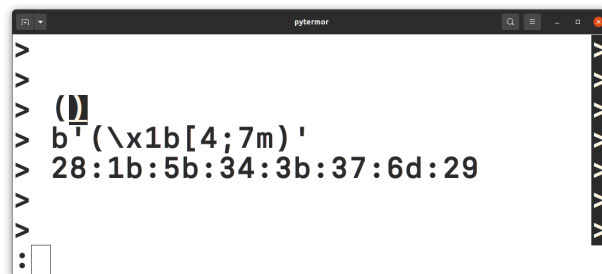
Important: SequenceSGR with zero params was specifically implemented to translate into an empty string and not into ESC [m, which would make sense, but also could be very entangling, as terminal emulators interpret that sequence as ESC [0m, which is *hard* reset sequence.

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

- `make_color_256()` will produce sequence operating in 256-colors mode (for a complete list see [Preset list](#));
- `make_color_rgb()` will create a sequence capable of setting the colors in True Color 16M mode (however, some terminal emulators doesn't support it).

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

```
1 from pytermor import SequenceSGR
2
3 seq = SequenceSGR(4, 7)
4 msg = f'({seq})'
5
6 print(msg + f'{SequenceSGR(0).assemble()}')
7 print(str(msg.assemble()))
8 print(msg.assemble().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 hexadecimal string representation.

1.3.4 SGR sequence structure

1. ESC is escape *control character*, which opens a control sequence (can also be written as `\x1b`, `\033` or `\e`).
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 SequenceSGR, SeqIndex
2
3 combined = SequenceSGR(1, 31) + SequenceSGR(4)
4 print(f'{combined}{combined[SeqIndex.RESET]}', str(combined).assemble())
```

1.3.6 Core API

Todo:

- *SequenceSGR* constructor
 - *SequenceSGR*.make_color_256()
 - *SequenceSGR*.make_color_rgb()
 - *Span* constructor
 - *Span*.init_explicit()
-

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 prsets.

There are two types of color palettes used in modern terminals – first one containing 16 colors (*Color16*), and second one consisting of 256 colors (*Color256*). 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, resetters

	Name	INT	SEQ	SPN	CLR	STY	Description
Meta							
	NOOP		V	V	V	V	No-operation; always assembled 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</i> mode]; use make_color_256 and make_color_rgb instead
	BG_COLOR_EXTENDED	48					Set background color [<i>indexed/RGB</i> mode]; use make_color_256 and make_color_rgb instead
	OVERLINED	53	V	V		V	Overline; <i>not widely supported</i>
Resetters							
	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 inverting
	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 Color16 presets


















































	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 Color256 presets

	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		#00ffa5	
	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 Xterm color palette

Actual colors of *default* palette depend on user's terminal settings, i.e. the result color of *Color16* 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).

Todo: (Verify) 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 *Color16* 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.

Todo: Approximation algorithm is as simple as iterating through all colors in the *lookup table* (which contains all possible ...)

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	001	002	003	004	005	006	007			
	#000000	#800000	#008000	#808000	#000080	#800080	#008080	#c0c0c0			
	008	009	010	011	012	013	014	015			
	#808080	#ff0000	#00ff00	#ffff00	#0000ff	#ff00ff	#00ffff	#ffffff			
016	022	028	034	040	046	082	076	070	064	058	052
#000000	#005f00	#008700	#00af00	#00d700	#00ff00	#5fff00	#5fd700	#5faf00	#5f8700	#5f5f00	#5f0000
017	023	029	035	041	047	083	077	071	065	059	053
#00005f	#005f5f	#00875f	#00af5f	#00d75f	#00ff5f	#5fff5f	#5fd75f	#5faf5f	#5f875f	#5f5f5f	#5f005f
018	024	030	036	042	048	084	078	072	066	060	054
#000087	#005f87	#008787	#00af87	#00d787	#00ff87	#5fff87	#5fd787	#5faf87	#5f8787	#5f5f87	#5f0087
019	025	031	037	043	049	085	079	073	067	061	055
#0000af	#005faf	#0087af	#00afaf	#00d7af	#00ffaf	#5fffaf	#5fd7af	#5fafaf	#5f87af	#5f5faf	#5f00af
020	026	032	038	044	050	086	080	074	068	062	056
#0000d7	#005fd7	#0087d7	#00afd7	#00dd7	#00ffd7	#5ffd7	#5fd7d7	#5fafd7	#5f87d7	#5f5fd7	#5f00d7
021	027	033	039	045	051	087	081	075	069	063	057
#0000ff	#005fff	#0087ff	#00afff	#00d7ff	#00ffff	#5fffff	#5fd7ff	#5fafff	#5f87ff	#5f5fff	#5f00ff
093	099	105	111	117	123	159	153	147	141	135	129
#8700ff	#875fff	#8787ff	#87afff	#87d7ff	#87ffff	#afffff	#afd7ff	#afafff	#af87ff	#af5fff	#af00ff
092	098	104	110	116	122	158	152	146	140	134	128
#8700d7	#875fd7	#8787d7	#87afd7	#87dd7	#87ffd7	#afffd7	#afd7d7	#afafd7	#af87d7	#af5fd7	#af00d7
091	097	103	109	115	121	157	151	145	139	133	127
#8700af	#875faf	#8787af	#87afaf	#87d7af	#87ffaf	#afffaf	#afd7af	#afafaf	#af87af	#af5faf	#af00af
090	096	102	108	114	120	156	150	144	138	132	126
#870087	#875f87	#878787	#87af87	#87d787	#87ff87	#afff87	#afd787	#afaf87	#af8787	#af5f87	#af0087
089	095	101	107	113	119	155	149	143	137	131	125
#87005f	#875f5f	#87875f	#87af5f	#87d75f	#87ff5f	#afff5f	#afd75f	#afaf5f	#af875f	#af5f5f	#af005f
088	094	100	106	112	118	154	148	142	136	130	124
#870000	#875f00	#878700	#87af00	#87d700	#87ff00	#afff00	#afd700	#afaf00	#af8700	#af5f00	#af0000
160	166	172	178	184	190	226	220	214	208	202	196
#d70000	#d75f00	#d78700	#dfa00	#dfd00	#dff00	#fff00	#ffd00	#ffa00	#ff8700	#ff5f00	#ff0000
161	167	173	179	185	191	227	221	215	209	203	197
#d7005f	#d75f5f	#d7875f	#dfa5f	#dfd5f	#dff5f	#fff5f	#ffd5f	#ffa5f	#ff875f	#ff5f5f	#ff005f
162	168	174	180	186	192	228	222	216	210	204	198
#d70087	#d75f87	#d78787	#dfa87	#dfd87	#dff87	#fff87	#ffd87	#ffa87	#ff8787	#ff5f87	#ff0087
163	169	175	181	187	193	229	223	217	211	205	199
#d700af	#d75faf	#d787af	#dfaaf	#dfdaf	#dffaf	#fffaf	#ffdaf	#ffaaf	#ff87af	#ff5faf	#ff00af
164	170	176	182	188	194	230	224	218	212	206	200
#d700d7	#d75fd7	#d787d7	#dafdf	#dfd7d7	#dff7d7	#fff7d7	#ffd7d7	#ffa7d7	#ff87d7	#ff5fd7	#ff00d7
165	171	177	183	189	195	231	225	219	213	207	201
#d700ff	#d75fff	#d787ff	#dafff	#dff7ff	#dff7ff	#fff7ff	#ffd7ff	#ffa7ff	#ff87ff	#ff5fff	#ff00ff
232	233	234	235	236	237	238	239	240	241	242	243
#080808	#121212	#1c1c1c	#262626	#303030	#3a3a3a	#444444	#4e4e4e	#585858	#626262	#6c6c6c	#767676
244	245	246	247	248	249	250	251	252	253	254	255
#808080	#8a8a8a	#949494	#9e9e9e	#a8a8a8	#b2b2b2	#bcbcbc	#c6c6c6	#d0d0d0	#dadada	#e4e4e4	#eeeeee

Fig. 2: Indexed mode palette

Sources

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

1.6 Named Colors collection

1.6.1 lisr

Todo: @TODO

1.7 String (and bytes) filters

1.7.1 filters

Todo: @TODO

1.8 Number formatters

Todo: The library contains @TODO

1.8.1 Auto-float formatter

1.8.2 Prefixed-unit formatter

1.8.3 Time delta formatter

```
1 import pytermor.utilnum
2 from pytermor import RendererManager, SgrRenderer
3 from pytermor.util import time_delta
4
5 seconds_list = [2, 10, 60, 2700, 32340, 273600, 4752000, 864000000]
6 max_len_list = [3, 6, 10]
7
8 for max_len in max_len_list:
9     formatter = pytermor.utilnum.registry.find_matching(max_len)
10
11 RendererManager.set_default(SgrRenderer)
12 for seconds in seconds_list:
13     for max_len in max_len_list:
14         formatter = pytermor.utilnum.registry.get_by_max_len(max_len)
```

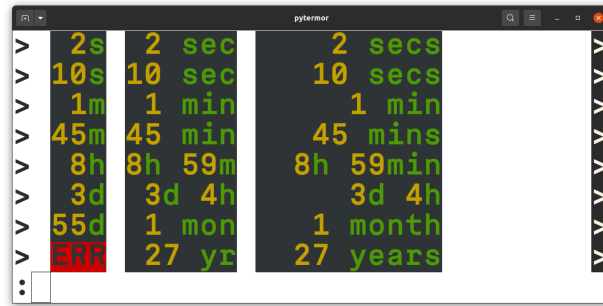
(continues on next page)

(continued from previous page)

```

15     print(formatter.format(seconds, True), end=' ')
16     print()

```



1.9 Documentation guidelines

(mostly as a reminder for myself)

- Basic types and built-in values should be surrounded with asterisks:

`*True*` → `True`

`*None*` → `None`

`*int*` → `int`

- Library classes, methods, etc. should be enclosed in single backticks in order to become a hyperlinks:

``SgrRenderer.render()`` → [`SgrRenderer.render\(\)`](#)

- Argument names and string literals that include escape sequences or their fragments should be wrapped in double backticks:

``arg1`` → `arg1`

``ESC [31m ESC [m`` → `ESC [31m ESC [m`

On the top of that, ESC control char should be padded with spaces for better readability. This also triggers automatic application of custom style for even more visual difference.

API REFERENCE

2.1 ansi

Module contains definitions for low-level ANSI escape sequences building. Can be used for creating a variety of sequences including:

- SGR sequences (text and background coloring, other text formatting and effects);
- CSI sequences (cursor management, selective screen clearing);
- OSC (Operating System Command) sequences (various system commands).

Important: blah-blah-blah low-level @TODO

The module doesn't distinguish "single-instruction" sequences from several ones merged together, e.g. `Style(fg='red', bold=True)` produces only one opening `SequenceSGR` instance:

```
>>> SequenceSGR(IntCode.BOLD, IntCode.RED).assemble()  
'[1;31m'
```

...although generally speaking it is two of them (ESC [1m and ESC [31m). However, the module can automatically match terminating sequences for any form of input SGRs and translate it to specified format.

XTerm Control Sequences

<https://invisible-island.net/xterm/ctlseqs/ctlseqs.html>

ECMA-48 specification

<https://www.ecma-international.org/publications-and-standards/standards/ecma-48/>

class `pytermor.ansi.IntCode(value)`

Bases: `IntEnum`

Complete or almost complete list of reliably working SGR param integer codes. Fully interchangeable with plain *int*. Suitable for `SequenceSGR` default constructor.

Note: `IntCode` predefined constants are omitted from documentation to avoid useless repeats and save space, as most of the time "next level" class `SeqIndex` is more appropriate, and on top of that, the constant names are literally the same for `SeqIndex` and `IntCode`.

classmethod `resolve(name)`

Parameters

name (*str*) –

Return type[IntCode](#)**class pytermor.ansi.SeqIndex**

Registry of static sequence presets.

BG_BLACK = <SGR[40]>

Set background color to 0x000000.

BG_BLUE = <SGR[44]>

Set background color to 0x000080.

BG_COLOR_OFF = <SGR[49]>

Reset background color.

BG_CYAN = <SGR[46]>

Set background color to 0x008080.

BG_GRAY = <SGR[100]>

Set background color to 0x808080.

BG_GREEN = <SGR[42]>

Set background color to 0x008000.

BG_HI_BLUE = <SGR[104]>

Set background color to 0x0000ff.

BG_HI_CYAN = <SGR[106]>

Set background color to 0x00ffff.

BG_HI_GREEN = <SGR[102]>

Set background color to 0x00ff00.

BG_HI_MAGENTA = <SGR[105]>

Set background color to 0xff00ff.

BG_HI_RED = <SGR[101]>

Set background color to 0xff0000.

BG_HI_WHITE = <SGR[107]>

Set background color to 0xffffffff.

BG_HI_YELLOW = <SGR[103]>

Set background color to 0xffff00.

BG_MAGENTA = <SGR[45]>

Set background color to 0x800080.

BG_RED = <SGR[41]>

Set background color to 0x800000.

BG_WHITE = <SGR[47]>

Set background color to 0xc0c0c0.

BG_YELLOW = <SGR[43]>

Set background color to 0x808000.

BLACK = <SGR[30]>

Set text color to 0x000000.

BLINK_FAST = <SGR[6]>

Set blinking to 150+ cpm (*not widely supported*).

BLINK_OFF = <SGR[25]>

Disable blinking.

BLINK_SLOW = <SGR[5]>

Set blinking to < 150 cpm.

BLUE = <SGR[34]>

Set text color to 0x000080.

BOLD = <SGR[1]>

Bold or increased intensity.

BOLD_DIM_OFF = <SGR[22]>

Disable BOLD and DIM attributes.

Special aspects... It's impossible to reliably disable them on a separate basis.

COLOR_OFF = <SGR[39]>

Reset foreground color.

CROSSLINED = <SGR[9]>

Strikethrough.

CROSSLINED_OFF = <SGR[29]>

Disable strikethrough.

CYAN = <SGR[36]>

Set text color to 0x008080.

DIM = <SGR[2]>

Faint, decreased intensity.

DOUBLE_UNDERLINED = <SGR[21]>

Double-underline. *On several terminals disables **BOLD** instead.*

GRAY = <SGR[90]>

Set text color to 0x808080.

GREEN = <SGR[32]>

Set text color to 0x008000.

HIDDEN = <SGR[8]>

Conceal characters (*not widely supported*).

HIDDEN_OFF = <SGR[28]>

Disable conecaling.

HI_BLUE = <SGR[94]>

Set text color to 0x0000ff.

HI_CYAN = <SGR[96]>

Set text color to 0x00ffff.

HI_GREEN = <SGR[92]>

Set text color to 0x00ff00.

HI_MAGENTA = <SGR[95]>

Set text color to 0xff00ff.

HI_RED = <SGR[91]>

Set text color to 0xff0000.

HI_WHITE = <SGR[97]>

Set text color to 0xffffff.

HI_YELLOW = <SGR[93]>

Set text color to 0xffff00.

HYPERLINK = <OSC[8]>

Create a hyperlink in the text (*supported by limited amount of terminals*). Note that for a working hyperlink you'll need two sequences, not just one.

See also:

[*make_hyperlink_part\(\)*](#) and [*assemble_hyperlink\(\)*](#).

INVERSED = <SGR[7]>

Swap foreground and background colors.

INVERSED_OFF = <SGR[27]>

Disable inverting.

ITALIC = <SGR[3]>

Italic (*not widely supported*).

ITALIC_OFF = <SGR[23]>

Disable italic.

MAGENTA = <SGR[35]>

Set text color to 0x800080.

OVERLINED = <SGR[53]>

Overline (*not widely supported*).

OVERLINED_OFF = <SGR[55]>

Disable overlining.

RED = <SGR[31]>

Set text color to 0x800000.

RESET = <SGR[0]>

Hard reset sequence.

UNDERLINED = <SGR[4]>

Underline.

UNDERLINED_OFF = <SGR[24]>

Disable underlining.

WHITE = <SGR[37]>

Set text color to 0xc0c0c0.

YELLOW = <SGR[33]>

Set text color to 0x808000.

class pytermor.ansi.Sequence(*params)

Bases: Sized, ABC

Abstract ancestor of all escape sequences.

assemble()

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

Return type

str

property params: t.List[int | str]

Return internal params as array.

class pytermor.ansi.SequenceCSI(terminator, short_name, *params)

Bases: [SequenceFe](#)

Class representing CSI-type ANSI escape sequence. All subtypes of this sequence start with ESC [.

Sequences of this type are used to control text formatting, change cursor position, erase screen and more.

```
>>> make_erase_in_line().assemble()
'[OK'
```

Parameters

- **terminator** –
- **short_name** –
- **params** –

assemble()

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

Return type

str

property params: t.List[int | str]

Return internal params as array.

class pytermor.ansi.SequenceFe(*params)

Bases: [Sequence](#), ABC

Wide range of sequence types that includes [CSI](#), [OSC](#) and more.

All subtypes of this sequence start with ESC plus ASCII byte from 0x40 to 0x5F (@, [, \,], _, ^ and capital letters A-Z).

assemble()

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

Return type

str

property params: t.List[int | str]

Return internal params as array.

class pytermor.ansi.SequenceOSC(*params)

Bases: [SequenceFe](#)

OSC-type sequence. Starts a control string for the operating system to use. Encoded as ESC], plus params separated by ;, and terminated with [SequenceST](#).

assemble()

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

Return type

str

property params: t.List[int | str]

Return internal params as array.

class pytermor.ansi.SequenceSGR(*args)

Bases: [SequenceCSI](#)

Class representing SGR-type escape sequence with varying amount of parameters. SGR sequences allow to change the color of text or/and terminal background (in 3 different color spaces) as well as set decorate text with italic style, underlining, overlining, cross-lining, making it bold or blinking etc.

When cast to *str*, as all other sequences, invokes [assemble\(\)](#) method and transforms into encoded control sequence string. It is possible to add of one SGR sequence to another, resulting in a new one with merged params (see examples).

Note: [SequenceSGR](#) with zero params was specifically implemented to translate into empty string and not into ESC [m, which would have made sense, but also would be entangling, as this sequence is the equivalent of ESC [0m – hard reset sequence. The empty-string-sequence is predefined at module level as [NOOP_SEQ](#).

```
>>> SequenceSGR(IntCode.HI_CYAN, 'underlined', 1)
<SGR[96,4,1]>
>>> SequenceSGR(31) + SequenceSGR(1) == SequenceSGR(31, 1)
True
```

Parameters

- **args** – Sequence params. Resulting param order is the same as an argument order. Each argument can be specified as:
 - *str* – any of [IntCode](#) names, case-insensitive
 - *int* – [IntCode](#) instance or plain integer
 - [SequenceSGR](#) instance (params will be extracted)
- **terminator** –
- **short_name** –
- **params** –

assemble()

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

Return type

str

property params: List[int]

Returns

Sequence params as integers or *IntCode* instances.

class pytermor.ansi.**SequenceST**(*params)

Bases: *SequenceFe*

String Terminator sequence (ST). Terminates strings in other control sequences. Encoded as ESC \ (0x1B 0x5C).

assemble()

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

Return type

str

property params: t.List[int | str]

Return internal params as array.

class pytermor.ansi.**UnderlinedCurlySequenceSGR**

Bases: *SequenceSGR*

Registered as a separate class, because this is the one and only SGR in the package, which is identified by “4:3” string (in contrast with all the other sequences entirely made of digits and semicolon separators).

assemble()

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

Return type

str

pytermor.ansi.**assemble_hyperlink**(url, label=None)

Parameters

- **url** (str) –
- **label** (Optional[str]) –

Example

ESC]8;;http://localhost ESC \Text ESC]8;; ESC \

Return type

str

pytermor.ansi.**enclose**(opening_seq, string)

Parameters

- **opening_seq** (*SequenceSGR*) –
- **string** (str) –

Returns

Return type

str

pytermor.ansi.**get_closing_seq**(opening_seq)

Parameters

- **opening_seq** (*SequenceSGR*) –

Returns**Return type**[SequenceSGR](#)

```
pytermor.ansi.make_color_256(code, bg=False)
```

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

Parameters

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

Example

```
ESC [38;5;141m
```

Return type[SequenceSGR](#)

```
pytermor.ansi.make_color_rgb(r, g, b, bg=False)
```

Wrapper for creation of [SequenceSGR](#) operating in True Color mode (16M). Valid values for *r*, *g* and *b* are in range of [0; 255]. This range linearly translates into [0x00; 0xFF] for each channel. The result value is composed as “0xRRGGBB”. For example, sequence with color of 0xFF3300 can be created with:

```
make_color_rgb(255, 51, 0)
```

Parameters

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

Example

```
ESC [38;2;255;51;0m
```

Return type[SequenceSGR](#)

```
pytermor.ansi.make_erase_in_line(mode=0)
```

Create EL (Erase in Line) sequence that erases a part of the line or the entire line. Cursor position does not change.

Parameters

mode (*int*) – Sequence operating mode.

- If set to 0, clear from cursor to the end of the line.
- If set to 1, clear from cursor to beginning of the line.
- If set to 2, clear the entire line.

Example

```
ESC [0K
```

Return type[SequenceCSI](#)

`pytermor.ansi.make_hyperlink_part(url=None)`

Parameters

`url` (*Optional*[*str*]) –

Example

ESC]8;;http://localhost ESC \

Return type

`SequenceOSC`

`pytermor.ansi.make_set_cursor_x_abs(x=1)`

Create CHA (Cursor Horizontal Absolute) sequence that sets cursor horizontal position, or column, to `x`.

Parameters

`x` (*int*) – New cursor horizontal position.

Example

ESC [1G

Return type

`SequenceCSI`

`pytermor.ansi.NOOP_SEQ = <SGR[NOP]>`

Special sequence in case you *have to* provide one or another SGR, but do not want any control sequences to be actually included in the output. `NOOP_SEQ.assemble()` returns empty string, `NOOP_SEQ.params` returns empty list.

```
>>> NOOP_SEQ.assemble()
""
>>> NOOP_SEQ.params
[]
```

2.2 color

exception `pytermor.color.ColorCodeConflictError(code, existing_color, new_color)`

Bases: `Exception`

exception `pytermor.color.ColorNameConflictError(tokens, existing_color, new_color)`

Bases: `Exception`

class `pytermor.color.ApxResult(color, distance)`

Bases: `Generic[CT]`

Approximation result.

color: `CT`

Found Color instance.

distance: `int`

Squared sRGB distance from this instance to the approximation target.

property distance_real: float

Actual distance from instance to target:

$$distance_{real} = \sqrt{distance}$$

class pytermor.color.Color16(*args, **kwargs)

Bases: Color

This variant of a Color operates within the most basic color set – **Xterm-16**. Represents basic color-setting SGRs with primary codes 30-37, 40-47, 90-97 and 100-107 (see [Color16 presets](#)).

Parameters

- **hex_value** – Color RGB value, e.g. 0x800000.
- **code_fg** – Int code for a foreground color setup, e.g. 30.
- **code_bg** – Int code for a background color setup. e.g. 40.
- **name** – Name of the color, e.g. “red”.
- **register** – If *True*, add color to registry for resolving by name.
- **index** – If *True*, add color to approximation index.
- **aliases** – Alternative color names (used in [resolve\(\)](#)).

classmethod approximate(hex_value, max_results=1)

Search for the colors nearest to hex_value and return the first max_results.

See

[color.approximate\(\)](#) for the details

Parameters

- **hex_value** (*int*) – Target RGB value.
- **max_results** (*int*) – Result limit.

Return type

List[ApxResult[CT]]

classmethod find_closest(hex_value)

Search and return nearest to hex_value color instance.

See

[color.find_closest\(\)](#) for the details

Parameters

hex_value (*int*) – Target RGB value.

Return type

CT

format_value(prefix='0x')

Format color value as “0xFFFFFFFF”.

Parameters

prefix (*str*) – Can be customized.

Return type

str

classmethod `get_by_code(code)`

Get a [Color16](#) instance with specified code. Only *foreground* (=text) colors are indexed, therefore it is impossible to look up for a [Color16](#) with given background color.

Parameters

code (*int*) – Foreground integer code to look up for (see [Color16 presets](#)).

Raises

KeyError – If no color with specified code is found.

Return type

[Color16](#)

classmethod `resolve(name)`

Case-insensitive search through registry contents.

See

[color.resolve\(\)](#) for the details

Parameters

name (*str*) – Color name to search for.

Return type

CT

to_hsv()

Wrapper around [hex_to_hsv\(\)](#) for concrete instance.

See

[hex_to_hsv\(\)](#) for the details

Return type

Tuple[float, float, float]

to_rgb()

Wrapper around [to_rgb\(\)](#) for concrete instance.

See

[to_rgb\(\)](#) for the details

Return type

Tuple[int, int, int]

to_sgr(bg, upper_bound=None)

Make an [SGR sequence](#) out of Color. Used by [SgrRenderer](#).

Parameters

- **bg** (*bool*) – Set to *True* if required SGR should change the background color, or *False* for the foreground (=text) color.
- **upper_bound** (*Optional*[*Type*[*Color*]]) – Required result Color type upper boundary, i.e., the maximum acceptable color class, which will be the basis for SGR being made. See [Color256.to_sgr\(\)](#) for the details.

Return type

[SequenceSGR](#)

to_tmux(bg)

Make a tmux markup directive, which will change the output color to this color's value (after tmux processes and prints it). Used by [TmuxRenderer](#).

Parameters

bg (*bool*) – Set to *True* if required tmux directive should change the background color, or *False* for the foreground (=text) color.

Return type

str

property code_bg: int

Int code for a background color setup. e.g. 40.

property code_fg: int

Int code for a foreground color setup, e.g. 30.

property hex_value: int

Color value, e.g. 0x3AEB0C.

property name: str | None

Color name, e.g. “navy-blue”.

class pytermor.color.Color256(*args, **kwargs)

Bases: Color

This variant of a Color operates within relatively modern **Xterm-256** indexed color table. Represents SGR complex codes 38;5;* and 48;5;* (see [Color256 presets](#)).

Parameters

- **hex_value** – Color RGB value, e.g. 0x5f0000.
- **code** – Int code for a color setup, e.g. 52.
- **name** – Name of the color, e.g. “dark-red”.
- **register** – If *True*, add color to registry for resolving by name.
- **index** – If *True*, add color to approximation index.
- **color16_equiv** – [Color16](#) counterpart (applies only to codes 0-15).
- **aliases** – Alternative color names (used in [resolve\(\)](#)).

classmethod approximate(hex_value, max_results=1)

Search for the colors nearest to hex_value and return the first max_results.

See

[color.approximate\(\)](#) for the details

Parameters

- **hex_value** (*int*) – Target RGB value.
- **max_results** (*int*) – Result limit.

Return type

List[[ApxResult](#)[CT]]

classmethod find_closest(hex_value)

Search and return nearest to hex_value color instance.

See

[color.find_closest\(\)](#) for the details

Parameters

hex_value (*int*) – Target RGB value.

Return type*CT***format_value**(*prefix*='0x')

Format color value as “0xFFFFFF”.

Parameters**prefix** (*str*) – Can be customized.**Return type***str***classmethod get_by_code**(*code*)Get a [Color256](#) instance with specified code (=position in the index).**Parameters****code** (*int*) – Color code to look up for (see [Color256 presets](#)).**Raises****KeyError** – If no color with specified code is found.**Return type**[Color256](#)**classmethod resolve**(*name*)

Case-insensitive search through registry contents.

See[color.resolve\(\)](#) for the details**Parameters****name** (*str*) – Color name to search for.**Return type***CT***to_hsv()**Wrapper around [hex_to_hsv\(\)](#) for concrete instance.**See**[hex_to_hsv\(\)](#) for the details**Return type***Tuple*[float, float, float]**to_rgb()**Wrapper around [to_rgb\(\)](#) for concrete instance.**See**[to_rgb\(\)](#) for the details**Return type***Tuple*[int, int, int]**to_sgr**(*bg*, *upper_bound*=None)Make an [SGR sequence](#) out of Color. Used by [SgrRenderer](#).

Each Color type represents one SGR type in the context of colors. For example, if *upper_bound* is set to [Color16](#), the resulting SGR will always be one of 16-color index table, even if the original color was of different type – it will be approximated just before the SGR assembling.

The reason for this is the necessity to provide a similar look for all users with different terminal settings/capabilities. When the library sees that user’s output device supports 256 colors only, it cannot assemble

True Color SGRs, because they will be ignored (if we are lucky), or displayed in a glitchy way, or mess up the output completely. The good news is that the process is automatic and in most cases the library will manage the transformations by itself. If it's not the case, the developer can correct the behaviour by overriding the renderers' output mode. See [SgrRenderer](#) and [OutputMode](#) docs.

Parameters

- **bg** (*bool*) – Set to *True* if required SGR should change the background color, or *False* for the foreground (=text) color.
- **upper_bound** (*Optional[Type[Color]]*) – Required result *Color* type upper boundary, i.e., the maximum acceptable color class, which will be the basis for SGR being made.

Return type

[SequenceSGR](#)

to_tmux(*bg*)

Make a tmux markup directive, which will change the output color to this color's value (after tmux processes and prints it). Used by [TmuxRenderer](#).

Parameters

- **bg** (*bool*) – Set to *True* if required tmux directive should change the background color, or *False* for the foreground (=text) color.

Return type

str

property code: int

Int code for a color setup, e.g. 52.

property hex_value: int

Color value, e.g. 0x3AEB0C.

property name: str | None

Color name, e.g. "navy-blue".

class `pytermor.color.ColorRGB(*args, **kwargs)`

Bases: *Color*

This variant of a *Color* operates within **Pytermor Named Colors**, unique collection of colors compiled from several known sources after careful selection (see [Named Colors collection](#)). However, it's not limited to aforementioned color list and can be easily extended.

Parameters

- **hex_value** – Color RGB value, e.g. 0x73a9c2.
- **name** – Name of the color, e.g. "moonstone-blue".
- **register** – If *True*, add color to registry for resolving by name.
- **index** – If *True*, add color to approximation index.
- **aliases** – Alternative color names (used in [resolve\(\)](#)).
- **variation_map** – Mapping {*int: str*}, where keys are hex values, and values are variation names.

classmethod `approximate(hex_value, max_results=1)`

Search for the colors nearest to **hex_value** and return the first **max_results**.

See

[color.approximate\(\)](#) for the details

Parameters

- **hex_value** (*int*) – Target RGB value.
- **max_results** (*int*) – Result limit.

Return type

List[*ApxResult*[*CT*]]

classmethod find_closest(*hex_value*)

Search and return nearest to *hex_value* color instance.

See

color.find_closest() for the details

Parameters

hex_value (*int*) – Target RGB value.

Return type

CT

format_value(*prefix*='0x')

Format color value as “0xFFFFFF”.

Parameters

prefix (*str*) – Can be customized.

Return type

str

classmethod resolve(*name*)

Case-insensitive search through registry contents.

See

color.resolve() for the details

Parameters

name (*str*) – Color name to search for.

Return type

CT

to_hsv()

Wrapper around *hex_to_hsv()* for concrete instance.

See

hex_to_hsv() for the details

Return type

Tuple[float, float, float]

to_rgb()

Wrapper around *to_rgb()* for concrete instance.

See

to_rgb() for the details

Return type

Tuple[int, int, int]

to_sgr(*bg*, *upper_bound*=None)

Make an *SGR sequence* out of Color. Used by *SgrRenderer*.

Parameters

- **bg** (*bool*) – Set to *True* if required SGR should change the background color, or *False* for the foreground (=text) color.
- **upper_bound** (*Optional[Type[Color]]*) – Required result *Color* type upper boundary, i.e., the maximum acceptable color class, which will be the basis for SGR being made. See [Color256.to_sgr\(\)](#) for the details.

Return type*SequenceSGR***to_tmux(bg)**

Make a tmux markup directive, which will change the output color to this color's value (after tmux processes and prints it). Used by [TmuxRenderer](#).

Parameters

- **bg** (*bool*) – Set to *True* if required tmux directive should change the background color, or *False* for the foreground (=text) color.

Return type*str***property base: CT | None**

Parent color for color variations. Empty for regular colors.

property hex_value: int

Color value, e.g. 0x3AEB0C.

property name: str | None

Color name, e.g. "navy-blue".

property variations: Dict[str, CT]

List of color variations. *Variation* of a color is a similar color with almost the same name, but with differing suffix. The main idea of variations is to provide a basis for fuzzy searching, which will return several results for one query; i.e., when the query matches a color with variations, the whole color family can be considered a match, which should increase searching speed.

pytermor.color.approximate(hex_value, color_type=None, max_results=1)

Search for nearest to *hex_value* colors of specified *color_type* and return the first *max_results* of them. If *color_type* is omitted, search for the closest [Color256](#) elements. This method is similar to the [find_closest\(\)](#), although they differ in some aspects:

- [approximate\(\)](#) can return more than one result;
- [approximate\(\)](#) returns not just a *Color* instance(s), but also a number equal to squared distance to the target color for each of them;
- [find_closest\(\)](#) caches the results, while [approximate\(\)](#) ignores the cache completely.

Parameters

- **hex_value** (*int*) – Target color RGB value.
- **color_type** (*Optional[Type[CT]]*) – Target color type ([Color16](#), [Color256](#) or [ColorRGB](#)).
- **max_results** (*int*) – Return no more than *max_results* items.

Returns

Pairs of closest *Color* instance(s) found with their distances to the target color, sorted by distance descending, i.e., element at index 0 is the closest color found, paired with its distance to the target; element with index 1 is second-closest color (if any) and corresponding distance value, etc.

Return type*List[ApxResult[CT]]*`pytermor.color.find_closest(hex_value, color_type=None)`

Search and return nearest to `hex_value` instance of specified `color_type`. If `color_type` is omitted, search for the closest `Color256` element.

Method is useful for finding applicable color alternatives if user's terminal is incapable of operating in more advanced mode. Usually it is done by the library automatically and transparently for both the developer and the end-user.

Note: This method caches the results, i.e., the same search query will from then onward result in the same return value without the necessity of iterating through the color index. If that's not applicable, use similar method `approximate()`, which is unaware of caching mechanism altogether.

Parameters

- **hex_value** (*int*) – Target color RGB value.
- **color_type** (*Optional[Type[CT]]*) – Target color type (`Color16`, `Color256` or `ColorRGB`).

Returns

Nearest to `hex_value` color instance of specified type.

Return type*CT*`pytermor.color.hex_to_hsv(hex_value)`

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

```
>>> hex_to_hsv(0x999999)
(0, 0.0, 0.6)
```

Parameters

hex_value (*int*) – RGB value.

Returns

H, S, V channel values correspondingly.

Return type*Tuple*[float, float, float]`pytermor.color.hex_to_rgb(hex_value)`

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

```
>>> hex_to_rgb(0x80ff80)
(128, 255, 128)
```

Parameters

hex_value (*int*) – RGB value.

Returns

R, G, B channel values correspondingly.

Return type

Tuple[int, int, int]

`pytermor.color.hsv_to_hex(h, s, v)`

Transforms HSV value in three-floats form (where $0 \leq h < 360$, $0 \leq s \leq 1$, and $0 \leq v \leq 1$) into an one-integer form 0xFFFFFF.

```
>>> hex(hsv_to_hex(90, 0.5, 0.5))
'0x608040'
```

Parameters

- **h** (*float*) – hue channel value.
- **s** (*float*) – saturation channel value.
- **v** (*float*) – value channel value.

Returns

RGB value.

Return type

int

`pytermor.color.hsv_to_rgb(h, s, v)`

Transforms HSV value in three-floats form (where $0 \leq h < 360$, $0 \leq s \leq 1$, and $0 \leq v \leq 1$) into RGB three-integer form ([0; 255], [0; 255], [0; 255]).

```
>>> hsv_to_rgb(270, 2/3, 0.75)
(128, 64, 192)
```

Parameters

- **h** (*float*) – hue channel value.
- **s** (*float*) – saturation channel value.
- **v** (*float*) – value channel value.

Returns

R, G, B channel values correspondingly.

Return type

Tuple[int, int, int]

`pytermor.color.resolve(name, color_type=None)`

Case-insensitive search through registry contents. Search is performed for `Color` instance of specified `color_type`, or in all three available registries if argument is omitted: first it will be performed in the registry of `Color16` class, then – in `Color256`, and, if previous two were unsuccessful, in the largest `ColorRGB` registry. Therefore, the return value could be any of these types.

Color names stored in registries as tokens, which allows to use any form of input and get the correct result regardless:

```
>>> resolve('deep-sky-blue-7')
<Color256[#23,005F5F,deep-sky-blue-7]>
>>> resolve('DEEP_SKY_BLUE_7')
<Color256[#23,005F5F,deep-sky-blue-7]>
>>> resolve('DeepSkyBlue7')
<Color256[#23,005F5F,deep-sky-blue-7]>
```

The only requirement is to split the words in any matter, so that tokenizer could distinguish the words from each other:

```
>>> try:
...     resolve('deepskyblue7')
... except LookupError as e:
...     print(e)
Color 'deepskyblue7' was not found in any of registries
```

Parameters

- **name** (*str*) – Color name to search for.
- **color_type** (*Optional[Type[CT]]*) – Target color type (*Color16*, *Color256* or *ColorRGB*).

Raises

LookupError – If nothing was found in either of registries.

Returns

Color instance with specified name.

Return type

CT

`pytermor.color.rgb_to_hex(r, g, b)`

Transforms RGB value in a three-integers form ([0; 255], [0; 255], [0; 255]) to an one-integer form 0xFFFFFF.

```
>>> hex(rgb_to_hex(0, 128, 0))
'0x8000'
```

Parameters

- **r** (*int*) – value of red channel.
- **g** (*int*) – value of green channel.
- **b** (*int*) – value of blue channel.

Returns

RGB value.

Return type

int

`pytermor.color.rgb_to_hsv(r, g, b)`

Transforms RGB value in a three-integers form ([0; 255], [0; 255], [0; 255]) to an HSV in three-floats form such as (0 ≤ h < 360, 0 ≤ s ≤ 1, and 0 ≤ v ≤ 1).

```
>>> rgb_to_hsv(0, 0, 255)
(240.0, 1.0, 1.0)
```

Parameters

- **r** (*int*) – value of red channel.
- **g** (*int*) – value of green channel.
- **b** (*int*) – value of blue channel.

Returns

H, S, V channel values correspondingly.

Return type

Tuple[float, float, float]

`pytermor.color.CT`

Any non-abstract Color type.

alias of `TypeVar('CT', Color16, Color256, ColorRGB)`

`pytermor.color.NOOP_COLOR = <_NoopColor[NOP]>`

Special Color instance always rendering into empty string.

2.3 common

exception `pytermor.common.ConflictError`

Bases: `Exception`

exception `pytermor.common.LogicError`

Bases: `Exception`

exception `pytermor.common.UserAbort`

Bases: `Exception`

exception `pytermor.common.UserCancel`

Bases: `Exception`

class `pytermor.common.Align(value)`

Bases: `str`, `Enum`

An enumeration.

`pytermor.common.StrType`

StrType in a method signature usually means that regular strings as well as *Renderable* implementations are supported, can be intermixed, and:

- return type will be *str* if and only if type of all arguments is *str*;
- otherwise return type will be *Renderable* – *str* arguments, if any, will be transformed into *Renderable* and concatenated.

alias of `TypeVar('StrType', bound=Union[str, Renderable])`

```
pytermor.common.T
    t.Any
    alias of TypeVar('T')
```

2.4 cval

Color preset list.

```
class pytermor.cval.CVAL

    AQUAMARINE_1 = <Color256[#122,87FFD7,aquamarine-1]>
    AQUAMARINE_2 = <Color256[#86,5FFFD7,aquamarine-2]>
    AQUAMARINE_3 = <Color256[#79,5FD7AF,aquamarine-3]>
    BLACK = <Color16[#30,000000?,black]>
    BLUE = <Color16[#34,000080?,blue]>
    BLUE_1 = <Color256[#21,0000FF,blue-1]>
    BLUE_2 = <Color256[#20,0000D7,blue-2]>
    BLUE_3 = <Color256[#19,0000AF,blue-3]>
    BLUE_VIOLET = <Color256[#57,5F00FF,blue-violet]>
    CADET_BLUE = <Color256[#73,5FAFAF,cadet-blue]>
    CADET_BLUE_2 = <Color256[#72,5FAF87,cadet-blue-2]>
    CHARTREUSE_1 = <Color256[#118,87FF00,chartreuse-1]>
    CHARTREUSE_2 = <Color256[#82,5FFF00,chartreuse-2]>
    CHARTREUSE_3 = <Color256[#112,87D700,chartreuse-3]>
    CHARTREUSE_4 = <Color256[#76,5FD700,chartreuse-4]>
    CHARTREUSE_5 = <Color256[#70,5FAF00,chartreuse-5]>
    CHARTREUSE_6 = <Color256[#64,5F8700,chartreuse-6]>
    CORNFLOWER_BLUE = <Color256[#69,5F87FF,cornflower-blue]>
    CORNSILK_1 = <Color256[#230,FFFFD7,cornsilk-1]>
    CYAN = <Color16[#36,008080?,cyan]>
    CYAN_1 = <Color256[#51,00FFFF,cyan-1]>
    CYAN_2 = <Color256[#50,00FFD7,cyan-2]>
    CYAN_3 = <Color256[#43,00D7AF,cyan-3]>
    DARK_BLUE = <Color256[#18,000087,dark-blue]>
```

```
DARK_CYAN = <Color256[#36,00AF87,dark-cyan]>
DARK_GOLDENROD = <Color256[#136,AF8700,dark-goldenrod]>
DARK_GREEN = <Color256[#22,005F00,dark-green]>
DARK_KHAKI = <Color256[#143,AFAF5F,dark-khaki]>
DARK_MAGENTA = <Color256[#91,8700AF,dark-magenta]>
DARK_MAGENTA_2 = <Color256[#90,870087,dark-magenta-2]>
DARK_OLIVE_GREEN_1 = <Color256[#192,D7FF87,dark-olive-green-1]>
DARK_OLIVE_GREEN_2 = <Color256[#191,D7FF5F,dark-olive-green-2]>
DARK_OLIVE_GREEN_3 = <Color256[#155,AFFF5F,dark-olive-green-3]>
DARK_OLIVE_GREEN_4 = <Color256[#113,87D75F,dark-olive-green-4]>
DARK_OLIVE_GREEN_5 = <Color256[#149,AFD75F,dark-olive-green-5]>
DARK_OLIVE_GREEN_6 = <Color256[#107,87AF5F,dark-olive-green-6]>
DARK_ORANGE = <Color256[#208,FF8700,dark-orange]>
DARK_ORANGE_2 = <Color256[#166,D75F00,dark-orange-2]>
DARK_ORANGE_3 = <Color256[#130,AF5F00,dark-orange-3]>
DARK_RED = <Color256[#88,870000,dark-red]>
DARK_RED_2 = <Color256[#52,5F0000,dark-red-2]>
DARK_SEA_GREEN_1 = <Color256[#158,AFFFD7,dark-sea-green-1]>
DARK_SEA_GREEN_2 = <Color256[#193,D7FFAF,dark-sea-green-2]>
DARK_SEA_GREEN_3 = <Color256[#157,AFFFAF,dark-sea-green-3]>
DARK_SEA_GREEN_4 = <Color256[#151,AFD7AF,dark-sea-green-4]>
DARK_SEA_GREEN_5 = <Color256[#115,87D7AF,dark-sea-green-5]>
DARK_SEA_GREEN_6 = <Color256[#150,AFD787,dark-sea-green-6]>
DARK_SEA_GREEN_7 = <Color256[#108,87AF87,dark-sea-green-7]>
DARK_SEA_GREEN_8 = <Color256[#71,5FAF5F,dark-sea-green-8]>
DARK_SEA_GREEN_9 = <Color256[#65,5F875F,dark-sea-green-9]>
DARK_SLATE_GRAY_1 = <Color256[#123,87FFFF,dark-slate-gray-1]>
DARK_SLATE_GRAY_2 = <Color256[#87,5FFFFFF,dark-slate-gray-2]>
DARK_SLATE_GRAY_3 = <Color256[#116,87D7D7,dark-slate-gray-3]>
DARK_TURQUOISE = <Color256[#44,00D7D7,dark-turquoise]>
DARK_VIOLET = <Color256[#128,AF00D7,dark-violet]>
```



```

DARK_VIOLET_2 = <Color256[#92,8700D7,dark-violet-2]>
DEEP_PINK_1 = <Color256[#199,FF00AF,deep-pink-1]>
DEEP_PINK_2 = <Color256[#198,FF0087,deep-pink-2]>
DEEP_PINK_3 = <Color256[#162,D70087,deep-pink-3]>
DEEP_PINK_4 = <Color256[#197,FF005F,deep-pink-4]>
DEEP_PINK_5 = <Color256[#161,D7005F,deep-pink-5]>
DEEP_PINK_6 = <Color256[#125,AF005F,deep-pink-6]>
DEEP_PINK_7 = <Color256[#89,87005F,deep-pink-7]>
DEEP_PINK_8 = <Color256[#53,5F005F,deep-pink-8]>
DEEP_SKY_BLUE_1 = <Color256[#39,00AFFF,deep-sky-blue-1]>
DEEP_SKY_BLUE_2 = <Color256[#38,00AFD7,deep-sky-blue-2]>
DEEP_SKY_BLUE_3 = <Color256[#32,0087D7,deep-sky-blue-3]>
DEEP_SKY_BLUE_4 = <Color256[#31,0087AF,deep-sky-blue-4]>
DEEP_SKY_BLUE_5 = <Color256[#25,005FAF,deep-sky-blue-5]>
DEEP_SKY_BLUE_6 = <Color256[#24,005F87,deep-sky-blue-6]>
DEEP_SKY_BLUE_7 = <Color256[#23,005F5F,deep-sky-blue-7]>
DODGER_BLUE_1 = <Color256[#33,0087FF,dodger-blue-1]>
DODGER_BLUE_2 = <Color256[#27,005FFF,dodger-blue-2]>
DODGER_BLUE_3 = <Color256[#26,005FD7,dodger-blue-3]>
GOLD_1 = <Color256[#220,FFD700,gold-1]>
GOLD_2 = <Color256[#178,D7AF00,gold-2]>
GOLD_3 = <Color256[#142,AFAF00,gold-3]>
GRAY = <Color16[#90,808080?,gray]>
GRAY_0 = <Color256[#16,000000,gray-0]>
GRAY_100 = <Color256[#231,FFFFFF,gray-100]>
GRAY_11 = <Color256[#234,1C1C1C,gray-11]>
GRAY_15 = <Color256[#235,262626,gray-15]>
GRAY_19 = <Color256[#236,303030,gray-19]>
GRAY_23 = <Color256[#237,3A3A3A,gray-23]>
GRAY_27 = <Color256[#238,444444,gray-27]>
GRAY_3 = <Color256[#232,080808,gray-3]>

```

```
GRAY_30 = <Color256[#239,4E4E4E,gray-30]>
GRAY_35 = <Color256[#240,585858,gray-35]>
GRAY_37 = <Color256[#59,5F5F5F,gray-37]>
GRAY_39 = <Color256[#241,626262,gray-39]>
GRAY_42 = <Color256[#242,6C6C6C,gray-42]>
GRAY_46 = <Color256[#243,767676,gray-46]>
GRAY_50 = <Color256[#244,808080,gray-50]>
GRAY_53 = <Color256[#102,878787,gray-53]>
GRAY_54 = <Color256[#245,8A8A8A,gray-54]>
GRAY_58 = <Color256[#246,949494,gray-58]>
GRAY_62 = <Color256[#247,9E9E9E,gray-62]>
GRAY_63 = <Color256[#139,AF87AF,gray-63]>
GRAY_66 = <Color256[#248,A8A8A8,gray-66]>
GRAY_69 = <Color256[#145,AFAFAF,gray-69]>
GRAY_7 = <Color256[#233,121212,gray-7]>
GRAY_70 = <Color256[#249,B2B2B2,gray-70]>
GRAY_74 = <Color256[#250,BCBCBC,gray-74]>
GRAY_78 = <Color256[#251,C6C6C6,gray-78]>
GRAY_82 = <Color256[#252,D0D0D0,gray-82]>
GRAY_84 = <Color256[#188,D7D7D7,gray-84]>
GRAY_85 = <Color256[#253,DADADA,gray-85]>
GRAY_89 = <Color256[#254,E4E4E4,gray-89]>
GRAY_93 = <Color256[#255,EEEEEE,gray-93]>
GREEN = <Color16[#32,008000?,green]>
GREEN_2 = <Color256[#46,00FF00,green-2]>
GREEN_3 = <Color256[#40,00D700,green-3]>
GREEN_4 = <Color256[#34,00AF00,green-4]>
GREEN_5 = <Color256[#28,008700,green-5]>
GREEN_YELLOW = <Color256[#154,FFFF00,green-yellow]>
HI_BLUE = <Color16[#94,0000FF?,hi-blue]>
HI_CYAN = <Color16[#96,00FFFF?,hi-cyan]>
```

```

HI_GREEN = <Color16[#92,00FF00?,hi-green]>
HI_MAGENTA = <Color16[#95,FF00FF?,hi-magenta]>
HI_RED = <Color16[#91,FF0000?,hi-red]>
HI_WHITE = <Color16[#97,FFFFFF?,hi-white]>
HI_YELLOW = <Color16[#93,FFFF00?,hi-yellow]>
HONEYDEW_2 = <Color256[#194,D7FFD7,honeydew-2]>
HOT_PINK = <Color256[#206,FF5FD7,hot-pink]>
HOT_PINK_2 = <Color256[#205,FF5FAF,hot-pink-2]>
HOT_PINK_3 = <Color256[#169,D75FAF,hot-pink-3]>
HOT_PINK_4 = <Color256[#168,D75F87,hot-pink-4]>
HOT_PINK_5 = <Color256[#132,AF5F87,hot-pink-5]>
INDIAN_RED_1 = <Color256[#203,FF5F5F,indian-red-1]>
INDIAN_RED_2 = <Color256[#204,FF5F87,indian-red-2]>
INDIAN_RED_3 = <Color256[#167,D75F5F,indian-red-3]>
INDIAN_RED_4 = <Color256[#131,AF5F5F,indian-red-4]>
KHAKI_1 = <Color256[#228,FFFF87,khaki-1]>
KHAKI_3 = <Color256[#185,D7D75F,khaki-3]>
LIGHT_CORAL = <Color256[#210,FF8787,light-coral]>
LIGHT_CYAN_1 = <Color256[#195,D7FFFF,light-cyan-1]>
LIGHT_CYAN_3 = <Color256[#152,AFD7D7,light-cyan-3]>
LIGHT_GOLDENROD_1 = <Color256[#227,FFFF5F,light-goldenrod-1]>
LIGHT_GOLDENROD_2 = <Color256[#222,FFD787,light-goldenrod-2]>
LIGHT_GOLDENROD_3 = <Color256[#186,D7D787,light-goldenrod-3]>
LIGHT_GOLDENROD_4 = <Color256[#221,FFD75F,light-goldenrod-4]>
LIGHT_GOLDENROD_5 = <Color256[#179,D7AF5F,light-goldenrod-5]>
LIGHT_GREEN = <Color256[#120,87FF87,light-green]>
LIGHT_GREEN_2 = <Color256[#119,87FF5F,light-green-2]>
LIGHT_PINK_1 = <Color256[#217,FFAFAF,light-pink-1]>
LIGHT_PINK_2 = <Color256[#174,D78787,light-pink-2]>
LIGHT_PINK_3 = <Color256[#95,875F5F,light-pink-3]>
LIGHT_SALMON_1 = <Color256[#216,FFAF87,light-salmon-1]>

```

```
LIGHT_SALMON_2 = <Color256[#173,D7875F,light-salmon-2]>
LIGHT_SALMON_3 = <Color256[#137,AF875F,light-salmon-3]>
LIGHT_SEA_GREEN = <Color256[#37,00AFAF,light-sea-green]>
LIGHT_SKY_BLUE_1 = <Color256[#153,AFD7FF,light-sky-blue-1]>
LIGHT_SKY_BLUE_2 = <Color256[#110,87AFD7,light-sky-blue-2]>
LIGHT_SKY_BLUE_3 = <Color256[#109,87AFAF,light-sky-blue-3]>
LIGHT_SLATE_BLUE = <Color256[#105,8787FF,light-slate-blue]>
LIGHT_SLATE_GRAY = <Color256[#103,8787AF,light-slate-gray]>
LIGHT_STEEL_BLUE_1 = <Color256[#189,D7D7FF,light-steel-blue-1]>
LIGHT_STEEL_BLUE_2 = <Color256[#147,AFAFFF,light-steel-blue-2]>
LIGHT_STEEL_BLUE_3 = <Color256[#146,AFAFD7,light-steel-blue-3]>
LIGHT_YELLOW_3 = <Color256[#187,D7D7AF,light-yellow-3]>
MAGENTA = <Color16[#35,800080?,magenta]>
MAGENTA_1 = <Color256[#201,FF00FF,magenta-1]>
MAGENTA_2 = <Color256[#200,FF00D7,magenta-2]>
MAGENTA_3 = <Color256[#163,D700AF,magenta-3]>
MAGENTA_4 = <Color256[#165,D700FF,magenta-4]>
MAGENTA_5 = <Color256[#164,D700D7,magenta-5]>
MAGENTA_6 = <Color256[#127,AF00AF,magenta-6]>
MEDIUM_ORCHID_1 = <Color256[#207,FF5FFF,medium-orchid-1]>
MEDIUM_ORCHID_2 = <Color256[#171,D75FFF,medium-orchid-2]>
MEDIUM_ORCHID_3 = <Color256[#134,AF5FD7,medium-orchid-3]>
MEDIUM_ORCHID_4 = <Color256[#133,AF5FAF,medium-orchid-4]>
MEDIUM_PURPLE_1 = <Color256[#141,AF87FF,medium-purple-1]>
MEDIUM_PURPLE_2 = <Color256[#135,AF5FFF,medium-purple-2]>
MEDIUM_PURPLE_3 = <Color256[#140,AF87D7,medium-purple-3]>
MEDIUM_PURPLE_4 = <Color256[#104,8787D7,medium-purple-4]>
MEDIUM_PURPLE_5 = <Color256[#98,875FD7,medium-purple-5]>
MEDIUM_PURPLE_6 = <Color256[#97,875FAF,medium-purple-6]>
MEDIUM_PURPLE_7 = <Color256[#60,5F5F87,medium-purple-7]>
MEDIUM_SPRING_GREEN = <Color256[#49,00FFAF,medium-spring-green]>
```

```
MEDIUM_TURQUOISE = <Color256[#80,5FD7D7,medium-turquoise]>
MEDIUM_VIOLET_RED = <Color256[#126,AF0087,medium-violet-red]>
MISTY_ROSE_1 = <Color256[#224,FFD7D7,misty-rose-1]>
MISTY_ROSE_3 = <Color256[#181,D7AFAF,misty-rose-3]>
NAVAJO_WHITE_1 = <Color256[#223,FFD7AF,navajo-white-1]>
NAVAJO_WHITE_3 = <Color256[#144,AFAF87,navajo-white-3]>
NAVY_BLUE = <Color256[#17,00005F,navy-blue]>
ORANGE_1 = <Color256[#214,FFAF00,orange-1]>
ORANGE_2 = <Color256[#172,D78700,orange-2]>
ORANGE_3 = <Color256[#94,875F00,orange-3]>
ORANGE_4 = <Color256[#58,5F5F00,orange-4]>
ORANGE_RED_1 = <Color256[#202,FF5F00,orange-red-1]>
ORCHID_1 = <Color256[#213,FF87FF,orchid-1]>
ORCHID_2 = <Color256[#212,FF87D7,orchid-2]>
ORCHID_3 = <Color256[#170,D75FD7,orchid-3]>
PALE_GREEN_1 = <Color256[#121,87FFAF,pale-green-1]>
PALE_GREEN_2 = <Color256[#156,AFFF87,pale-green-2]>
PALE_GREEN_3 = <Color256[#114,87D787,pale-green-3]>
PALE_GREEN_4 = <Color256[#77,5FD75F,pale-green-4]>
PALE_TURQUOISE_1 = <Color256[#159,AFFFFFF,pale-turquoise-1]>
PALE_TURQUOISE_4 = <Color256[#66,5F8787,pale-turquoise-4]>
PALE_VIOLET_RED_1 = <Color256[#211,FF87AF,pale-violet-red-1]>
PINK_1 = <Color256[#218,FFAFD7,pink-1]>
PINK_3 = <Color256[#175,D787AF,pink-3]>
PLUM_1 = <Color256[#219,FFAFFF,plum-1]>
PLUM_2 = <Color256[#183,D7AFFF,plum-2]>
PLUM_3 = <Color256[#176,D787D7,plum-3]>
PLUM_4 = <Color256[#96,875F87,plum-4]>
PURPLE = <Color256[#129,AF00FF,purple]>
PURPLE_2 = <Color256[#93,8700FF,purple-2]>
PURPLE_3 = <Color256[#56,5F00D7,purple-3]>
```

```
PURPLE_4 = <Color256[#55,5F00AF,purple-4]>
PURPLE_6 = <Color256[#54,5F0087,purple-6]>
RED = <Color16[#31,800000?,red]>
RED_1 = <Color256[#196,FF0000,red-1]>
RED_3 = <Color256[#160,D70000,red-3]>
RED_4 = <Color256[#124,AF0000,red-4]>
ROSY_BROWN = <Color256[#138,AF8787,rosy-brown]>
ROYAL_BLUE_1 = <Color256[#63,5F5FFF,royal-blue-1]>
SALMON_1 = <Color256[#209,FF875F,salmon-1]>
SANDY_BROWN = <Color256[#215,FFAF5F,sandy-brown]>
SEA_GREEN_1 = <Color256[#85,5FFFAF,sea-green-1]>
SEA_GREEN_2 = <Color256[#84,5FFF87,sea-green-2]>
SEA_GREEN_3 = <Color256[#78,5FD787,sea-green-3]>
SEA_GREEN_4 = <Color256[#83,5FFF5F,sea-green-4]>
SKY_BLUE_1 = <Color256[#117,87D7FF,sky-blue-1]>
SKY_BLUE_2 = <Color256[#111,87AFFF,sky-blue-2]>
SKY_BLUE_3 = <Color256[#74,5FAFD7,sky-blue-3]>
SLATE_BLUE_1 = <Color256[#99,875FFF,slate-blue-1]>
SLATE_BLUE_2 = <Color256[#62,5F5FD7,slate-blue-2]>
SLATE_BLUE_3 = <Color256[#61,5F5FAF,slate-blue-3]>
SPRING_GREEN_1 = <Color256[#48,00FF87,spring-green-1]>
SPRING_GREEN_2 = <Color256[#47,00FF5F,spring-green-2]>
SPRING_GREEN_3 = <Color256[#41,00D75F,spring-green-3]>
SPRING_GREEN_4 = <Color256[#29,00875F,spring-green-4]>
SPRING_GREEN_5 = <Color256[#35,00AF5F,spring-green-5]>
SPRING_GREEN_6 = <Color256[#42,00D787,spring-green-6]>
STEEL_BLUE = <Color256[#67,5F87AF,steel-blue]>
STEEL_BLUE_1 = <Color256[#81,5FD7FF,steel-blue-1]>
STEEL_BLUE_2 = <Color256[#75,5FAFFF,steel-blue-2]>
STEEL_BLUE_3 = <Color256[#68,5F87D7,steel-blue-3]>
TAN = <Color256[#180,D7AF87,tan]>
```

```

THISTLE_1 = <Color256[#225,FFD7FF,thistle-1]>
THISTLE_3 = <Color256[#182,D7AFD7,thistle-3]>
TURQUOISE_2 = <Color256[#45,00D7FF,turquoise-2]>
TURQUOISE_4 = <Color256[#30,008787,turquoise-4]>
VIOLET = <Color256[#177,D787FF,violet]>
WHEAT_1 = <Color256[#229,FFFFAF,wheat-1]>
WHEAT_4 = <Color256[#101,87875F,wheat-4]>
WHITE = <Color16[#37,C0C0C0?,white]>
YELLOW = <Color16[#33,808000?,yellow]>
YELLOW_1 = <Color256[#226,FFFF00,yellow-1]>
YELLOW_2 = <Color256[#190,D7FF00,yellow-2]>
YELLOW_3 = <Color256[#184,D7D700,yellow-3]>
YELLOW_4 = <Color256[#106,87AF00,yellow-4]>
YELLOW_5 = <Color256[#148,AFD700,yellow-5]>
YELLOW_6 = <Color256[#100,878700,yellow-6]>

```

2.5 renderer

Module with output formatters. Default global renderer type is *SgrRenderer*.

Setting up a rendering mode can be accomplished in several ways:

- By using general-purpose functions `text.render()` and `text.echo()` – both have an argument `renderer` (preferable; introduced in pytermor 2.x).
- Method `RendererManager.set_default()` sets the default renderer globally. After that calling `text.render()` will automatically invoke a said renderer and apply the required formatting (that is, if `renderer` argument is left empty).
- Alternatively, you can use renderer's own instance method `render()` directly and avoid messing up with the manager: `HtmlRenderer.render()` (not recommended and possibly will be deprecated in future versions).

Generally speaking, if you need to invoke a custom renderer just once, it's convenient to use the first method for this matter, and use the second one in all the other cases.

On the contrary, if there is a necessity to use more than one renderer alternately, it's better to avoid using the global one at all, and just instantiate and invoke both renderers independently.

TL;DR

To unconditionally print formatted message to standard output, do something like this:

```
>>> from pytermor import render, RendererManager, Styles
>>> RendererManager.set_default_format_always()
>>> render('Warning: AAAA', Styles.WARNING)
'[33mWarning: AAAA[39m'
```

class pytermor.renderer.AbstractRenderer

Bases: ABC

Renderer interface.

clone(*args, **kwargs)

Make a copy of the renderer with the same setup.

Return type

self

abstract render(string, fmt=<Style[NOP]>)

Apply colors and attributes described in *fmt* argument to *string* and return the result. Output format depends on renderer's class, which defines the implementation.

Parameters

- **string** (*str*) – String to format.
- **fmt** (*Color* / *Style*) – Style or color to apply. If *fmt* is a *Color* instance, it is assumed to be a foreground color.

Returns

String with formatting applied, or without it, depending on renderer settings.

Return type

str

property is_caching_allowed: bool

Returns

True if caching of renderer's results makes any sense and *False* otherwise.

abstract property is_format_allowed: bool

Returns

True if renderer is set up to use the formatting and will do it on invocation, and *False* otherwise.

class pytermor.renderer.HtmlRenderer

Bases: *AbstractRenderer*

Translate *Styles* attributes into a rudimentary HTML markup. All the formatting is inlined into *style* attribute of the `` elements. Can be optimized by extracting the common styles as CSS classes and referencing them by DOM elements instead.

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

clone(*args, **kwargs)

Make a copy of the renderer with the same setup.

Return type

self

render(*string*, *fmt*=<Style[NOP]>)

Apply colors and attributes described in *fmt* argument to *string* and return the result. Output format depends on renderer's class, which defines the implementation.

Parameters

- **string** (*str*) – String to format.
- **fmt** (*Color* / *Style*) – Style or color to apply. If *fmt* is a *Color* instance, it is assumed to be a foreground color.

Returns

String with formatting applied, or without it, depending on renderer settings.

Return type

str

property is_caching_allowed: bool**Returns**

True if caching of renderer's results makes any sense and *False* otherwise.

property is_format_allowed: bool**Returns**

Always *True*, because the capabilities of the terminal have nothing to do with HTML markup meant for web-browsers.

class pytermor.renderer.NoOpRenderer

Bases: [AbstractRenderer](#)

Special renderer type that does nothing with the input string and just returns it as is. Often used as a default argument value (along with similar “NoOps” like [NOOP_STYLE](#), [NOOP_COLOR](#) etc.)

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

clone(*args, **kwargs)

Make a copy of the renderer with the same setup.

Return type

self

render(*string*, *fmt*=<Style[NOP]>)

Return the *string* argument untouched, don't mind the *fmt*.

Parameters

- **string** (*str*) – String to format ignore.
- **fmt** (*Color* / *Style*) – Style or color to appl discard.

Return type

str

property is_caching_allowed: bool**Returns**

True if caching of renderer's results makes any sense and *False* otherwise.

property `is_format_allowed`: `bool`

Returns

Nothing to apply → nothing to allow, thus the returned value is always *False*.

class `pytermor.renderer.OutputMode(value)`

Bases: `Enum`

Determines what types of SGR sequences are allowed to use in the output.

AUTO = `'auto'`

Lets the renderer select the most suitable mode by itself. See [SgrRenderer](#) constructor documentation for the details.

NO_ANSI = `'no_ansi'`

The renderer discards all color and format information completely.

TRUE_COLOR = `'true_color'`

RGB color mode. Does not apply restrictions to color rendering.

XTERM_16 = `'xterm_16'`

16-colors mode. Enforces the renderer to approximate all color types to [Color16](#) and render them as basic mode selection SGR sequences (ESC [31m, ESC [42m etc). See `Color.approximate()` for approximation algorithm details.

XTERM_256 = `'xterm_256'`

256-colors mode. Allows the renderer to use either [Color16](#) or [Color256](#) (but RGB will be approximated to 256-color palette).

class `pytermor.renderer.RendererManager`

Class for global renderer setup.

classmethod `get_default()`

Get global renderer instance ([SgrRenderer](#), or the one provided earlier with `set_default()`).

Return type

[AbstractRenderer](#)

classmethod `set_default(renderer=None)`

Select a global renderer.

```
>>> RendererManager.set_default(SgrRendererDebugger(OutputMode.XTERM_16))
>>> render('text', Style(fg='red'))
'([31m)text([39m)'
```

Parameters

renderer ([AbstractRenderer](#) / `t.Type[AbstractRenderer]`) – Default renderer to use globally. Calling this method without arguments will result in library default renderer [SgrRenderer](#) being set as default.

All the methods with the `renderer` argument (e.g., `text.render()`) will use the global default one if said argument is omitted or set to *None*.

You can specify either the renderer class, in which case manager will instantiate it with the default parameters, or provide already instantiated and set up renderer, which will be registered as global.

classmethod `set_default_format_always()`

Shortcut for forcing all control sequences to be present in the output of a global renderer.

Note that it applies only to the renderer that is set up as default at the moment of calling this method, i.e., all previously created instances, as well as the ones that will be created afterwards, are unaffected.

classmethod `set_default_format_never()`

Shortcut for disabling all output formatting of a global renderer.

```
class pytermor.renderer.SgrRenderer(output_mode=OutputMode.AUTO, io=<_io.TextIOWrapper
                                name='<stdout>' mode='w' encoding='utf-8'>)
```

Bases: [AbstractRenderer](#)

Default renderer invoked by `Text.render()`. Transforms `Color` instances defined in `style` into ANSI control sequence bytes and merges them with input string. Type of resulting [SequenceSGR](#) depends on type of `Color` instances in `style` argument and current output mode of the renderer.

1. [ColorRGB](#) can be rendered as True Color sequence, 256-color sequence or 16-color sequence depending on specified [OutputMode](#).
2. [Color256](#) can be rendered as 256-color sequence or 16-color sequence.
3. [Color16](#) will be rendered as 16-color sequence.
4. Nothing of the above will happen and all formatting will be discarded completely if output device is not a terminal emulator or if the developer explicitly set up the renderer to do so ([OutputMode.NO_ANSI](#)).

Renderer approximates RGB colors to closest **indexed** colors if terminal doesn't support RGB output. In case terminal doesn't support even 256 colors, it falls back to 16-color palette and picks closest samples again the same way. See [OutputMode](#) documentation for exact mappings.

```
>>> SgrRenderer(OutputMode.XTERM_256).render('text', Styles.WARNING_LABEL)
'[1;33mtext[22;39m'
>>> SgrRenderer(OutputMode.NO_ANSI).render('text', Styles.WARNING_LABEL)
'text'
```

Parameters

output_mode ([OutputMode](#)) – SGR output mode to use. Valid values are listed in [OutputMode](#) enum.

With [OutputMode.AUTO](#) the renderer will first check if the output device is a terminal emulator, and use [OutputMode.NO_ANSI](#) when it is not. Otherwise, the renderer will read `TERM` environment variable and follow these rules:

- [OutputMode.NO_ANSI](#) if `TERM` is set to `xterm`.
- [OutputMode.XTERM_16](#) if `TERM` is set to `xterm-color`.
- [OutputMode.XTERM_256](#) in all other cases.

Special case is when `TERM` equals to `xterm-256color` **and** `COLORTERM` is either `truecolor` or `24bit`, then [OutputMode.TRUE_COLOR](#) will be used.

`clone()`

Make a copy of the renderer with the same setup.

Return type

self

render(*string*, *fmt*=<Style[NOP]>)

Apply colors and attributes described in *fmt* argument to *string* and return the result. Output format depends on renderer's class, which defines the implementation.

Parameters

- **string** (*str*) – String to format.
- **fmt** (*Color* / *Style*) – Style or color to apply. If *fmt* is a *Color* instance, it is assumed to be a foreground color.

Returns

String with formatting applied, or without it, depending on renderer settings.

Return type

str

property is_caching_allowed: bool

Returns

True if caching of renderer's results makes any sense and *False* otherwise.

property is_format_allowed: bool

Returns

True if renderer is set up to use the formatting and will do it on invocation, and *False* otherwise.

class pytermor.renderer.**SgrRendererDebugger**(*output_mode*=*OutputMode.AUTO*)

Bases: *SgrRenderer*

Subclass of regular *SgrRenderer* with two differences – instead of rendering the proper ANSI escape sequences it renders them with ESC character replaced by “”, and encloses the whole sequence into ‘()’ for visual separation.

Can be used for debugging of assembled sequences, because such a transformation reliably converts a control sequence into a harmless piece of bytes completely ignored by the terminals.

```
>>> SgrRendererDebugger(OutputMode.XTERM_16).render('text', Style(fg='red',  
↪ bold=True))  
'([1;31m)text([22;39m)'
```

clone()

Make a copy of the renderer with the same setup.

Return type

self

render(*string*, *fmt*=<Style[NOP]>)

Apply colors and attributes described in *fmt* argument to *string* and return the result. Output format depends on renderer's class, which defines the implementation.

Parameters

- **string** (*str*) – String to format.
- **fmt** (*Color* / *Style*) – Style or color to apply. If *fmt* is a *Color* instance, it is assumed to be a foreground color.

Returns

String with formatting applied, or without it, depending on renderer settings.

Return type

str

set_format_always()

Force all control sequences to be present in the output.

set_format_auto()Reset the force formatting flag and let the renderer decide by itself (see [SgrRenderer](#) docs for the details).**set_format_never()**

Force disabling of all output formatting.

property is_caching_allowed: bool**Returns***True* if caching of renderer's results makes any sense and *False* otherwise.**property is_format_allowed: bool****Returns***True* if renderer is set up to use the formatting and will do it on invocation, and *False* otherwise.**class pytermor.renderer.TmuxRenderer**Bases: [AbstractRenderer](#)Translates [Styles](#) attributes into [tmux-compatible](#) markup. [tmux](#) is a commonly used terminal multiplexer.

```
>>> TmuxRenderer().render('text', Style(fg='blue', bold=True))
'#[fg=blue bold]text#[fg=default nobold]'
```

clone(*args, **kwargs)

Make a copy of the renderer with the same setup.

Return type

self

render(string, fmt=<Style[NOP]>)Apply colors and attributes described in *fmt* argument to *string* and return the result. Output format depends on renderer's class, which defines the implementation.**Parameters**

- **string** (*str*) – String to format.
- **fmt** (*Color* / *Style*) – Style or color to apply. If *fmt* is a *Color* instance, it is assumed to be a foreground color.

Returns

String with formatting applied, or without it, depending on renderer settings.

Return type

str

property is_caching_allowed: bool**Returns***True* if caching of renderer's results makes any sense and *False* otherwise.

property is_format_allowed: `bool`

Returns

Always *True*, because tmux markup can be used without regard to the type of output device and its capabilities – all the dirty work will be done by the multiplexer itself.

2.6 style

```
class pytermor.style.Style(parent=None, fg=None, bg=None, blink=None, bold=None, crosslined=None,
                           dim=None, double_underlined=None, inversed=None, italic=None,
                           overlined=None, underlined=None, class_name=None)
```

style

Create a new `Style()`. Both `fg` and `bg` can be specified as:

1. `Color` instance or library preset;
2. **str** – name of any of these presets, case-insensitive;
3. **int** – color value in hexadecimal RGB format;
4. *None* – the color will be unset.

Inheritance `parent -> child` works this way:

- If an argument in child's constructor is empty (*None*), take value from `parent`'s corresponding attribute.
- If an argument in child's constructor is *not* empty (`True`, `False`, `Color` etc.), use it as child's attribute.

Note: Both empty (i.e., *None*) attributes of type `Color` after initialization will be replaced with special constant `NOOP_COLOR`, which behaves like there was no color defined, and at the same time makes it safer to work with nullable color-type variables.

```
>>> Style(fg='green', bold=True)
<Style[fg=<Color16[#32,008000?,green]>,bg=<_NoopColor[NOP]>,bold]>
>>> Style(bg=0x0000ff)
<Style[fg=<_NoopColor[NOP]>,bg=<ColorRGB[0000FF]>]>
>>> Style(fg='DeepSkyBlue1', bg='gray3')
<Style[fg=<Color256[#39,00AFFF,deep-sky-blue-1]>,bg=<Color256[#232,080808,gray-3]>]>
```

Parameters

- **parent** (`Style`) – Style to copy attributes without value from.
- **fg** (`Color` | `int` | `str`) – Foreground (i.e., text) color.
- **bg** (`Color` | `int` | `str`) – Background color.
- **blink** (`bool`) – Blinking effect; *supported by limited amount of Renderers*.
- **bold** (`bool`) – Bold or increased intensity.
- **crosslined** (`bool`) – Strikethrough.
- **dim** (`bool`) – Faint, decreased intensity.
- **double_underlined** (`bool`) – Faint, decreased intensity.
- **inversed** (`bool`) – Swap foreground and background colors.

- **italic** (*bool*) – Italic.
- **overlined** (*bool*) – Overline.
- **underlined** (*bool*) – Underline.
- **class_name** (*str*) – Arbitrary string used by some `_get_renderers`, e.g. by `HtmlRenderer`.

autopick_fg()

Pick `fg_color` depending on `bg_color`. Set `fg_color` to either 3% gray (almost black) if background is bright, or to 80% gray (bright gray) if it is dark. If background is `None`, do nothing.

Todo: check if there is a better algorithm, because current thinks text on #000080 should be black

Returns

self

Return type

Style

flip()

Swap foreground color and background color. :return: self

Return type

Style

class pytermor.style.Styles

Some ready-to-use styles. Can be used as examples.

`pytermor.style.NOOP_STYLE = <Style[NOP]>`

Special style passing the text through without any modifications.

2.7 text

class pytermor.text.FixedString(*string=""*, *fmt=<Style[NOP]>*, *align=Align.LEFT*, *width=0*, *pad_left=0*, *pad_right=0*, *overflow_char=None*)

Bases: *String*

class pytermor.text.FrozenText(*string=""*, *fmt=<Style[NOP]>*, *close_this=True*, *close_prev=False*)

Bases: *Renderable*

class pytermor.text.Renderable

Bases: *Sized*, *ABC*

Renderable abstract class. Can be inherited when the default style overlaps resolution mechanism implemented in *Text* is not good enough.

class pytermor.text.String(*string=""*, *fmt=<Style[NOP]>*)

Bases: *Renderable*

class pytermor.text.Text(*string=""*, *fmt=<Style[NOP]>*, *close_this=True*, *close_prev=False*)

Bases: *FrozenText*

```
pytermor.text.echo(string="", fmt=<Style[NOP]>, renderer=None, parse_template=False, nl=True,
                    file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>, flush=True,
                    wrap=False, indent_first=0, indent_subseq=0)
```

Parameters

- **string** (*StrType* | *t.Iterable[StrType]*) –
- **fmt** (*Color* | *Style*) –
- **renderer** (*AbstractRenderer*) –
- **parse_template** (*bool*) –
- **nl** (*bool*) –
- **file** (*t.IO*) –
- **flush** (*bool*) –
- **wrap** (*bool* | *int*) –
- **indent_first** (*int*) –
- **indent_subseq** (*int*) –

```
pytermor.text.render(string="", fmt=<Style[NOP]>, renderer=None, parse_template=False)
```

Parameters

- **string** (*StrType* | *t.Iterable[StrType]*) –
- **fmt** (*Color* | *Style*) –
- **renderer** (*AbstractRenderer*) –
- **parse_template** (*bool*) –

Returns

Return type

str | *t.List[str]*

2.8 utilmisc

```
pytermor.utilmisc.confirm(attempts=1, default=False, keymap=None, prompt=None, quiet=False,
                           required=False)
```

Ensure the next action is manually confirmed by user. Print the terminal prompt with **prompt** text and wait for a keypress. Return *True* if user pressed **Y** and *False* in all the other cases (by default).

Valid keys are **Y** and **N** (case insensitive), while all the other keys and combinations are considered invalid, and will trigger the return of the default value, which is *False* if not set otherwise. In other words, by default the user is expected to press either **Y** or **N**, and if that's not the case, the confirmation request will be automatically failed.

Ctrl+C instantly aborts the confirmation process regardless of attempts count and raises *UserAbort*.

Example keymap (default one):

```
keymap = {"y": True, "n": False}
```

Parameters

- **attempts** (*int*) – Set how many times the user is allowed to perform the input before auto-cancellation (or auto-confirmation) will occur. 1 means there will be only one attempt, the first one. When set to -1, allows to repeat the input infinitely.
- **default** (*bool*) – Default value that will be returned when user presses invalid key (e.g. Backspace, Ctrl+Q etc.) and his **attempts** counter decreases to 0. Setting this to *True* effectively means that the user's only way to deny the request is to press N or Ctrl+C, while all the other keys are treated as Y.
- **keymap** (*Optional[Mapping[str, bool]]*) – Key to result mapping.
- **prompt** (*Optional[str]*) – String to display before each input attempt. Default is: "Press Y to continue, N to cancel, Ctrl+C to abort: "
- **quiet** (*bool*) – If set to *True*, suppress all messages to stdout and work silently.
- **required** (*bool*) – If set to *True*, raise *UserCancel* or *UserAbort* when user rejects to confirm current action. If set to *False*, do not raise any exceptions, just return *False*.

Returns

True if there was a confirmation by user's input or automatically, *False* otherwise.

Raises

UserAbort

Raises

UserCancel

Return type

bool

`pytermor.utilmisc.get_preferable_wrap_width(force_width=None)`

Return preferable terminal width for comfort reading of wrapped text.

Return type

int

`pytermor.utilmisc.get_terminal_width(default=80, padding=2)`

Return current terminal width with an optional "safety buffer".

Return type

int

`pytermor.utilmisc.total_size(o, handlers=None, verbose=False)`

Returns the approximate memory footprint of an object and all of its contents.

Automatically finds the contents of the following builtin containers and their subclasses: tuple, list, deque, dict, set and frozenset. To search other containers, add handlers to iterate over their contents:

```
handlers = {SomeContainerClass: iter,
            OtherContainerClass: OtherContainerClass.get_elements}
```

Return type

int

`pytermor.utilmisc.wait_key()`

Wait for a key press on the console and return it.

Raises

EOFError

Return type

t.AnyStr | None

2.9 utilnum

```
class pytermor.utilnum.PrefixedUnitFormatter(max_value_len, truncate_frac=None, unit=None,
                                             unit_separator=None, mcoef=None, prefixes=None,
                                             prefix_zero_idx=None, parent=None)
```

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.

Parameters

- **max_value_len** (*int*) –
- **truncate_frac** (*bool*) –
- **unit** (*str*) –
- **unit_separator** (*str*) –
- **mcoef** (*float*) –
- **prefixes** (*List[str | None]*) –
- **prefix_zero_idx** (*int*) – Index of prefix which will be used as default, i.e. without multiplying coefficients.
- **parent** (*PrefixedUnitFormatter*) –

New in version 1.7.

```
format(value, unit=None, join=True)
```

Parameters

- **value** (*float*) – Input value
- **unit** (*str*) – Unit override
- **join** (*bool*) – Return the result as a string if set to *True*, or as a (num, sep, unit) tuple otherwise.

Returns

Formatted value

Return type

str | Tuple[str, str, str]

```
property max_len: int
```

Returns

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

```
class pytermor.utilnum.TimeDeltaFormatter(units, allow_negative, unit_separator=None,
                                          plural_suffix=None, overflow_msg='OVERFLOW')
```

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"
```

Parameters

- **units** (*List[TimeUnit]*) –
- **allow_negative** (*bool*) –
- **unit_separator** (*str*) –
- **plural_suffix** (*str*) –
- **overflow_msg** (*str*) –

```
format(seconds, always_max_len=False)
```

Pretty-print difference between two moments in time.

Parameters

- **seconds** (*float*) – Input value.
- **always_max_len** (*bool*) – 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.

Return type

str

```
format_raw(seconds)
```

Pretty-print difference between two moments in time, do not replace the output with “OVERFLOW” warning message.

Parameters

- seconds** (*float*) – Input value.

Returns

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

Return type

str | None

```
property max_len: int
```

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

Returns

Maximum possible output string length.

```
class pytermor.utilnum.TimeUnit(name: 'str', in_next: 'int' = None, custom_short: 'str' = None,
                                collapsible_after: 'int' = None, overflow_after: 'int' = None)
```

```
pytermor.utilnum.format_auto_float(value, req_len, allow_exponent_notation=True)
```

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** (*float*) – Value to format
- **req_len** (*int*) – Required output string length
- **allow_exponent_notation** (*bool*) – Enable/disable exponent form.

Returns

Formatted string of required length

Raises

ValueError –

Return type

str

New in version 1.7.

```
pytermor.utilnum.format_si_binary(value, unit='b', join=True)
```

Format value as binary size (bytes, kbytes, Mbytes), max result length is 8 chars: 5 for value plus 3 for default unit, prefix and separator. Base is 1024. Unit can be customized.

```
>>> format_si_binary(1010) # 1010 b < 1 kb
'1010 b'
>>> format_si_binary(1080)
```

(continues on next page)

(continued from previous page)

```
'1 kb'
>>> format_si_binary(45200)
'44 kb'
>>> format_si_binary(1.258 * pow(10, 6), 'bps')
'1 Mbps'
```

Parameters

- **value** (*float*) – Input value in bytes.
- **unit** (*str*) – Value unit, printed right after the prefix.
- **join** (*bool*) – Return the result as a string if set to *True*, or as a (num, sep, unit) tuple otherwise.

Returns

Formatted string with SI-prefix if necessary.

Return type

str | Tuple[str, str, str]

New in version 2.0.

pytermor.utilnum.**format_si_metric**(value, unit='m', join=True)

Format value as meters with SI-prefixes, max result length is 7 chars: 4 for value plus 3 for default unit, prefix and separator. Base is 1000. Unit can be customized. Suitable for formatting any SI unit with values from approximately 10^{-27} to 10^{27} .

```
>>> format_si_metric(1010, 'm²')
'1.01 km²'
>>> format_si_metric(0.0319, 'g')
'31.9 mg'
>>> format_si_metric(1213531546, 'W') # great scott
'1.21 GW'
>>> format_si_metric(1.26e-9, 'eV')
'1.26 neV'
```

Parameters

- **value** (*float*) – Input value (unitless).
- **unit** (*str*) – Value unit, printed right after the prefix.
- **join** (*bool*) – Return the result as a string if set to *True*, or as a (num, sep, unit) tuple otherwise.

Returns

Formatted string with SI-prefix if necessary.

Return type

str | Tuple[str, str, str]

New in version 2.0.

pytermor.utilnum.**format_thousand_sep**(value, separator='')

Returns input value with integer part split 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'
```

Parameters

- **value** (*int* | *float*) –
- **separator** (*str*) –

Return type

str

`pytermor.utilnum.format_time_delta(seconds, max_len=None)`

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** (*float*) – Value to format
- **max_len** (*Optional[int]*) – Maximum output string length (total)

Returns

Formatted string

Return type

str

`pytermor.utilnum.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.utilnum.PREFIX_ZERO_SI = 8`

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

`pytermor.utilnum._formatter_si_binary = PrefixedUnitFormatter`

Configuration example, used by `format_si_metric`.

While being similar to `_formatter_si_metric`, this formatter differs in one aspect. Given a variable with default value = 995, formatting it's value results in “995 b”. After increasing it by 20 we'll have 1015, but it's

still not enough to become a kilobyte – so returned value will be “1015 b”. Only after one more increase (at 1024 and more) the value will be in a form of “1.00 kb”.

So, in this case `max_value_len` must be at least 5 (not 4), because it’s a minimum requirement for formatting values from 1023 to -1023.

Total maximum length is `max_value_len + 3 = 8` (+3 is from separator, unit and prefix, assuming all of them have 1-char width).

`pytermor.utilnum._formatter_si_metric = PrefixedUnitFormatter`

Configuration example, used by `format_si_binary`.

`max_value_len` must be at least 4, because it’s a minimum requirement for formatting values from 999 to -999. Next number to 999 is 1000, which will be formatted as “1k”.

Total maximum length is `max_value_len + 3`, which is 7 (+3 is from separator, unit and prefix, assuming all of them have 1-char width). Without unit (default) it’s 6.

2.10 utilstr

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. Also includes several Python Standard Library methods rewritten for correct work with strings containing control sequences.

class `pytermor.utilstr.BytesHexPrinter(char_per_line=32)`

Bases: `GenericPrinter[bytes]`

str/bytes as byte hex codes, grouped by 4

Listing 1: Example output

```
0000  0A 20 32 31 36 20 20 20  E2 94 82 20 20 75 70 6C  |a
0010  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |a
```

class `pytermor.utilstr.CsiStringReplacer(repl="")`

Bases: `StringReplacer`

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

Parameters

repl – Replacement, can contain regexp groups (see `apply_filters()`).

class `pytermor.utilstr.GenericFilter`

Bases: `Generic[IT, OT]`, `ABC`

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.

__call__(s)

Can be used instead of `apply()`

Return type

OT

apply(inp, extra=None)

Apply the filter to input *str* or *bytes*.

Parameters

- **inp** (*IT*) – input string
- **extra** (*Optional[Any]*) – additional options

Returns

transformed string; the type can match the input type, as well as be different – that depends on filter type.

Return type

OT

class pytermor.utilstr.**GenericPrinter**(*char_per_line*)

Bases: *GenericFilter*[*IT*, *str*], *ABC*

apply(*inp*, *extra=None*)

Apply the filter to input *str* or *bytes*.

Parameters

- **inp** (*IT*) – input string
- **extra** (*Optional[PrinterExtra]*) – additional options

Returns

transformed string; the type can match the input type, as well as be different – that depends on filter type.

Return type

str

class pytermor.utilstr.**GenericStringPrinter**(*char_per_line*)

Bases: *GenericPrinter*[*str*], *ABC*

class pytermor.utilstr.**NonPrintablesOmniVisualizer**(*override=None*)

Bases: *OmniMapper*

Input type: *str*, *bytes*. Replace every whitespace character with ..

class pytermor.utilstr.**NonPrintablesStringVisualizer**(*keep_newlines=True*)

Bases: *StringMapper*

Input type: *str*. Replace every whitespace character with “.”, except newlines. Newlines are kept and get prepended with same char by default, but this behaviour can be disabled with *keep_newlines = False*.

```
>>> NonPrintablesStringVisualizer().apply('A B C')
'A..B..C'
>>> apply_filters('1. D'+os.linesep+'2. L ', NonPrintablesStringVisualizer(keep_
↪newlines=False))
'1..D2..L..'
```

Parameters

keep_newlines – When *True*, transform newline characters into “\n”, or into just “.” otherwise.

class pytermor.utilstr.**OmniMapper**(*override=None*)

Bases: *GenericFilter*[*IT*, *IT*]

Input type: *str*, *bytes*. Abstract mapper. Replaces every character found in map keys to corresponding map value. Map should be a dictionary of this type: *dict[int, str|bytes|None]*; moreover, length of *str/bytes* must be strictly 1 character (ASCII codepage). If there is a necessity to map Unicode characters, *StringMapper* should be used instead.


```
>>> OmniMapper({0x20: ' '}).apply(b'abc def ghi')
b'abc.def.ghi'
```

For mass mapping it is better to subclass [OmniMapper](#) and override two methods – `_get_default_keys` and `_get_default_replacer`. In this case you don't have to manually compose a replacement map with every character you want to replace.

Parameters

override – a dictionary with mappings: keys must be *ints*, values must be either a single-char *strs* or *bytes*, or *None*.

See

[NonPrintablesOmniVisualizer](#)

apply(*inp*, *extra=None*)

Apply the filter to input *str* or *bytes*.

Parameters

- **inp** (*IT*) – input string
- **extra** (*Optional[Any]*) – additional options

Returns

transformed string; the type can match the input type, as well as be different – that depends on filter type.

Return type

IT

class pytermor.utilstr.**OmniSanitizer**(*repl=b'.'*)

Bases: [OmniMapper](#)

Input type: *str*, *bytes*. Replace every control character and every non-ASCII character (0x80-0xFF) with “.”, or with specified char. Note that the replacement should be a single ASCII character, because *Omni*- filters are designed to work with *str* inputs and *bytes* inputs on equal terms.

Parameters

repl – Value to replace control/non-ascii characters with. Should be strictly 1 character long.

class pytermor.utilstr.**PrinterExtra**(*label: 'str'*)

class pytermor.utilstr.**SgrStringReplacer**(*repl=""*)

Bases: [StringReplacer](#)

Find all SGR seqs (e.g. ESC[1;4m) and replace with given string. More specific version of *CsiReplacer*.

Parameters

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

class pytermor.utilstr.**StringHexPrinter**(*char_per_line=16*)

Bases: [GenericStringPrinter](#)

str as byte hex codes (UTF-8), grouped by characters

Listing 2: Example output

0056	45 4D 20 43 50 55	20	4F 56 48 20 4E	45 3E 0A 20	E
0072	20 20 20 20 20 20 E29482		20 20 20 20 20	20 20 20 20	_
0088	20 20 20 20 37 20	2B	30 20 20 20 20 CE94	20 32 68	_
0104	20 33 33 6D 20 20	20 EFAA8F	20 2D 35 20 C2B0	43 20 20	_

```
class pytermor.utilstr.StringMapper(override=None)
```

```
    Bases: OmniMapper[str]
```

```
    a
```

```
    apply(inp, extra=None)
```

```
        Apply the filter to input str or bytes.
```

Parameters

- **inp** (*str*) – input string
- **extra** (*Optional* [*Any*]) – additional options

Returns

transformed string; the type can match the input type, as well as be different – that depends on filter type.

Return type

str

```
class pytermor.utilstr.StringReplacer(pattern, repl)
```

```
    Bases: GenericFilter[str, str]
```

```
    .
```

```
    apply(inp, extra=None)
```

```
        Apply the filter to input str or bytes.
```

Parameters

- **inp** (*str*) – input string
- **extra** (*Optional* [*Any*]) – additional options

Returns

transformed string; the type can match the input type, as well as be different – that depends on filter type.

Return type

str

```
class pytermor.utilstr.StringUcpPrinter(char_per_line=16)
```

```
    Bases: GenericStringPrinter
```

```
    str as Unicode codepoints
```

Todo: venv/lib/python3.8/site-packages/pygments/lexers/hexdump.py

Listing 3: Example output

```

56 |U+ 45 4d 20 43 50 55 20 4f 56 48 20 4e 45 3e 0a 20 | EM_CPU_OVH_NE>
72 |U+ 20 20 20 20 20 20 2502 20 20 20 20 20 20 20 20 | |
88 |U+ 20 20 20 20 37 20 2b 30 20 20 20 20 394 20 32 68 | 7+02h
104 |U+ 20 33 33 6d 20 20 20 fa8f 20 2d 35 20 b0 43 20 20 | 33m-5°C

```

`pytermor.utilstr.apply_filters(string, *args)`

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):

```

>>> from pytermor import SeqIndex
>>> apply_filters(f'{SeqIndex.RED}test{SeqIndex.COLOR_OFF}', SgrStringReplacer(r'E}
↪'))
'E[31mtestE[39m'

```

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

Parameters

- **string** (*IT*) – String to filter.
- **args** (`Union[OmniFilter, Type[OmniFilter]]`) – `OmniFilter` instance(s) or `OmniFilter` type(s).

Returns

Filtered `s`.

Return type

OT

`pytermor.utilstr.center_sgr(s, width, fillchar=' ', actual_len=None)`

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-

Return type

`str`

`pytermor.utilstr.distribute_padded(values, max_len, pad_before=False, pad_after=False)`

Todo: todo

Parameters

- **values** (`List[StrType]`) –
- **max_len** (`int`) –
- **pad_before** (`bool`) –
- **pad_after** (`bool`) –

Return type*StrType*`pytermor.utilstr.dump(data, label=None, max_len_shift=None)`

Todo:

- format selection
 - special handling of one-line input
 - squash repeating lines
-

Return type`str | None``pytermor.utilstr.ljust_sgr(s, width, fillchar=' ', actual_len=None)`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).

Return type`str``pytermor.utilstr.rjust_sgr(s, width, fillchar=' ', actual_len=None)`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).

Return type`str``pytermor.utilstr.wrap_sgr(raw_input, width, indent_first=0, indent_subseq=0)`

A workaround to make standard library `textwrap.wrap()` more friendly to an SGR-formatted strings.

The main idea is

Parameters

- `raw_input` (`str | list[str]`) –
- `width` (`int`) –

Return type`str`

CHANGELOG

3.1 v2.23-dev

- Extracted *resolve*, *approximate*, *find_closest* from `Color` class to module-level.
- As well as color transform functions.
- Add missing *hsv_to_rgb* function.
- `GenericPrniter` and *OmniMapper*.
- *StringHexPrinter* and *StringUcpPrinter*.
- *SgrRenderer* support for custom I/O streams.
- *FrozenText* class.

3.2 v2.18-dev

- *cval* autobuild.
- `ArgCountError` migrated from `es7s/core`.
- `black` code style.
- Add `OmniHexPrinter` and `chunk()` helper.
- Typehinting.
- Disabled automatic rendering of *echo()* and *render()*.

3.3 v2.14-dev

Dec 22

- *confirm()* helper command.
- `EscapeSequenceStringReplacer` filter.
- `examples/terminal_benchmark` script.
- `StringFilter` and `OmniFilter` classes.
- Docs design fixes.
- Minor core improvements.

- Tests for *color* module.
- RGB and variations full support.

3.4 v2.6-dev

Nov 22

- Got rid of `Span` class.
- Rewrite of *color* module.
- Changes in `ConfigurableRenderer.force_styles` logic.
- *Text* nesting.
- `TemplateEngine` implementation.
- Package reorganizing.

3.5 v2.2-dev

Oct 22

- Named colors list.
- *Renderable* interface.
- Color config.
- *TmuxRenderer*
- `wait_key()` input helper.

3.6 v2.1-dev

Aug 22

- Color presets.
- More unit tests for formatters.

3.7 v2.0-dev

Jul 22

- 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.

3.8 v1.8

Jun 22

- `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).

3.9 v1.7.4

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

3.10 v1.7.3

May 22

- Added `span.BG_BLACK` format.

3.11 v1.7.2

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

3.12 v1.7.1

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

3.13 v1.7

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

3.14 v1.6.2

- Excluded tests dir from distribution package.

3.15 v1.6.1

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

3.16 v1.5

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

3.17 v1.4

- `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.

3.18 v1.3.2

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

3.19 v1.3.1

- Interface revisioning.

3.20 v1.2.1

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

3.21 v1.2

- EmptySequenceSGR and EmptyFormat classes.

3.22 v1.1

Apr 22

- Autoformat feature.

3.23 v1.0

- First public version.

3.24 v0.90

Mar 22

- First commit.

This project uses Semantic Versioning – <https://semver.org> (*starting from 2.0*)

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.ansi`, [23](#)
- `pytermor.color`, [31](#)
- `pytermor.common`, [42](#)
- `pytermor.cval`, [43](#)
- `pytermor.renderer`, [51](#)
- `pytermor.style`, [58](#)
- `pytermor.text`, [59](#)
- `pytermor.utilmisc`, [60](#)
- `pytermor.utilnum`, [62](#)
- `pytermor.utilstr`, [67](#)

Symbols

`__call__()` (*pytermor.utilstr.GenericFilter* method), 67
`_formatter_si_binary` (in module *pytermor.utilnum*), 66
`_formatter_si_metric` (in module *pytermor.utilnum*), 67

A

AbstractRenderer (class in *pytermor.renderer*), 52
Align (class in *pytermor.common*), 42
`apply()` (*pytermor.utilstr.GenericFilter* method), 67
`apply()` (*pytermor.utilstr.GenericPrinter* method), 68
`apply()` (*pytermor.utilstr.OmniMapper* method), 69
`apply()` (*pytermor.utilstr.StringMapper* method), 70
`apply()` (*pytermor.utilstr.StringReplacer* method), 70
`apply_filters()` (in module *pytermor.utilstr*), 71
`approximate()` (in module *pytermor.color*), 38
`approximate()` (*pytermor.color.Color16* class method), 32
`approximate()` (*pytermor.color.Color256* class method), 34
`approximate()` (*pytermor.color.ColorRGB* class method), 36
ApxResult (class in *pytermor.color*), 31
AQUAMARINE_1 (*pytermor.cval.CVAL* attribute), 43
AQUAMARINE_2 (*pytermor.cval.CVAL* attribute), 43
AQUAMARINE_3 (*pytermor.cval.CVAL* attribute), 43
`assemble()` (*pytermor.ansi.Sequence* method), 27
`assemble()` (*pytermor.ansi.SequenceCSI* method), 27
`assemble()` (*pytermor.ansi.SequenceFe* method), 27
`assemble()` (*pytermor.ansi.SequenceOSC* method), 28
`assemble()` (*pytermor.ansi.SequenceSGR* method), 28
`assemble()` (*pytermor.ansi.SequenceST* method), 29
`assemble()` (*pytermor.ansi.UnderlinedCurlySequenceSGR* method), 29
`assemble_hyperlink()` (in module *pytermor.ansi*), 29
AUTO (*pytermor.renderer.OutputMode* attribute), 54
`autopick_fg()` (*pytermor.style.Style* method), 59

B

base (*pytermor.color.ColorRGB* property), 38
BG_BLACK (*pytermor.ansi.SeqIndex* attribute), 24

BG_BLUE (*pytermor.ansi.SeqIndex* attribute), 24
BG_COLOR_OFF (*pytermor.ansi.SeqIndex* attribute), 24
BG_CYAN (*pytermor.ansi.SeqIndex* attribute), 24
BG_GRAY (*pytermor.ansi.SeqIndex* attribute), 24
BG_GREEN (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_BLUE (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_CYAN (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_GREEN (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_MAGENTA (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_RED (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_WHITE (*pytermor.ansi.SeqIndex* attribute), 24
BG_HI_YELLOW (*pytermor.ansi.SeqIndex* attribute), 24
BG_MAGENTA (*pytermor.ansi.SeqIndex* attribute), 24
BG_RED (*pytermor.ansi.SeqIndex* attribute), 24
BG_WHITE (*pytermor.ansi.SeqIndex* attribute), 24
BG_YELLOW (*pytermor.ansi.SeqIndex* attribute), 24
BLACK (*pytermor.ansi.SeqIndex* attribute), 24
BLACK (*pytermor.cval.CVAL* attribute), 43
BLINK_FAST (*pytermor.ansi.SeqIndex* attribute), 24
BLINK_OFF (*pytermor.ansi.SeqIndex* attribute), 25
BLINK_SLOW (*pytermor.ansi.SeqIndex* attribute), 25
BLUE (*pytermor.ansi.SeqIndex* attribute), 25
BLUE (*pytermor.cval.CVAL* attribute), 43
BLUE_1 (*pytermor.cval.CVAL* attribute), 43
BLUE_2 (*pytermor.cval.CVAL* attribute), 43
BLUE_3 (*pytermor.cval.CVAL* attribute), 43
BLUE_VIOLET (*pytermor.cval.CVAL* attribute), 43
BOLD (*pytermor.ansi.SeqIndex* attribute), 25
BOLD_DIM_OFF (*pytermor.ansi.SeqIndex* attribute), 25
BytesHexPrinter (class in *pytermor.utilstr*), 67

C

CADET_BLUE (*pytermor.cval.CVAL* attribute), 43
CADET_BLUE_2 (*pytermor.cval.CVAL* attribute), 43
`center_sgr()` (in module *pytermor.utilstr*), 71
CHARTREUSE_1 (*pytermor.cval.CVAL* attribute), 43
CHARTREUSE_2 (*pytermor.cval.CVAL* attribute), 43
CHARTREUSE_3 (*pytermor.cval.CVAL* attribute), 43
CHARTREUSE_4 (*pytermor.cval.CVAL* attribute), 43
CHARTREUSE_5 (*pytermor.cval.CVAL* attribute), 43
CHARTREUSE_6 (*pytermor.cval.CVAL* attribute), 43

`clone()` (*pytermor.renderer.AbstractRenderer method*), 52
`clone()` (*pytermor.renderer.HtmlRenderer method*), 52
`clone()` (*pytermor.renderer.NoOpRenderer method*), 53
`clone()` (*pytermor.renderer.SgrRenderer method*), 55
`clone()` (*pytermor.renderer.SgrRendererDebugger method*), 56
`clone()` (*pytermor.renderer.TmuxRenderer method*), 57
`code` (*pytermor.color.Color256 property*), 36
`code_bg` (*pytermor.color.Color16 property*), 34
`code_fg` (*pytermor.color.Color16 property*), 34
`color` (*pytermor.color.ApxResult attribute*), 31
`Color16` (*class in pytermor.color*), 32
`Color256` (*class in pytermor.color*), 34
`COLOR_OFF` (*pytermor.ansi.SeqIndex attribute*), 25
`ColorCodeConflictError`, 31
`ColorNameConflictError`, 31
`ColorRGB` (*class in pytermor.color*), 36
`confirm()` (*in module pytermor.utilmisc*), 60
`ConflictError`, 42
`CORNFLOWER_BLUE` (*pytermor.cval.CVAL attribute*), 43
`CORNSILK_1` (*pytermor.cval.CVAL attribute*), 43
`CROSSLINED` (*pytermor.ansi.SeqIndex attribute*), 25
`CROSSLINED_OFF` (*pytermor.ansi.SeqIndex attribute*), 25
`CsiStringReplacer` (*class in pytermor.utilstr*), 67
`CT` (*in module pytermor.color*), 42
`CVAL` (*class in pytermor.cval*), 43
`CYAN` (*pytermor.ansi.SeqIndex attribute*), 25
`CYAN` (*pytermor.cval.CVAL attribute*), 43
`CYAN_1` (*pytermor.cval.CVAL attribute*), 43
`CYAN_2` (*pytermor.cval.CVAL attribute*), 43
`CYAN_3` (*pytermor.cval.CVAL attribute*), 43

D

`DARK_BLUE` (*pytermor.cval.CVAL attribute*), 43
`DARK_CYAN` (*pytermor.cval.CVAL attribute*), 43
`DARK_GOLDENROD` (*pytermor.cval.CVAL attribute*), 44
`DARK_GREEN` (*pytermor.cval.CVAL attribute*), 44
`DARK_KHAKI` (*pytermor.cval.CVAL attribute*), 44
`DARK_MAGENTA` (*pytermor.cval.CVAL attribute*), 44
`DARK_MAGENTA_2` (*pytermor.cval.CVAL attribute*), 44
`DARK_OLIVE_GREEN_1` (*pytermor.cval.CVAL attribute*), 44
`DARK_OLIVE_GREEN_2` (*pytermor.cval.CVAL attribute*), 44
`DARK_OLIVE_GREEN_3` (*pytermor.cval.CVAL attribute*), 44
`DARK_OLIVE_GREEN_4` (*pytermor.cval.CVAL attribute*), 44
`DARK_OLIVE_GREEN_5` (*pytermor.cval.CVAL attribute*), 44
`DARK_OLIVE_GREEN_6` (*pytermor.cval.CVAL attribute*), 44
`DARK_ORANGE` (*pytermor.cval.CVAL attribute*), 44
`DARK_ORANGE_2` (*pytermor.cval.CVAL attribute*), 44
`DARK_ORANGE_3` (*pytermor.cval.CVAL attribute*), 44
`DARK_RED` (*pytermor.cval.CVAL attribute*), 44
`DARK_RED_2` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_1` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_2` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_3` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_4` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_5` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_6` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_7` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_8` (*pytermor.cval.CVAL attribute*), 44
`DARK_SEA_GREEN_9` (*pytermor.cval.CVAL attribute*), 44
`DARK_SLATE_GRAY_1` (*pytermor.cval.CVAL attribute*), 44
`DARK_SLATE_GRAY_2` (*pytermor.cval.CVAL attribute*), 44
`DARK_SLATE_GRAY_3` (*pytermor.cval.CVAL attribute*), 44
`DARK_TURQUOISE` (*pytermor.cval.CVAL attribute*), 44
`DARK_VIOLET` (*pytermor.cval.CVAL attribute*), 44
`DARK_VIOLET_2` (*pytermor.cval.CVAL attribute*), 44
`DEEP_PINK_1` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_2` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_3` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_4` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_5` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_6` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_7` (*pytermor.cval.CVAL attribute*), 45
`DEEP_PINK_8` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_1` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_2` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_3` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_4` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_5` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_6` (*pytermor.cval.CVAL attribute*), 45
`DEEP_SKY_BLUE_7` (*pytermor.cval.CVAL attribute*), 45
`DIM` (*pytermor.ansi.SeqIndex attribute*), 25
`distance` (*pytermor.color.ApxResult attribute*), 31
`distance_real` (*pytermor.color.ApxResult property*), 31
`distribute_padded()` (*in module pytermor.utilstr*), 71
`DODGER_BLUE_1` (*pytermor.cval.CVAL attribute*), 45
`DODGER_BLUE_2` (*pytermor.cval.CVAL attribute*), 45
`DODGER_BLUE_3` (*pytermor.cval.CVAL attribute*), 45
`DOUBLE_UNDERLINED` (*pytermor.ansi.SeqIndex attribute*), 25
`dump()` (*in module pytermor.utilstr*), 72

E

`echo()` (*in module pytermor.text*), 59
`enclose()` (*in module pytermor.ansi*), 29

F

`find_closest()` (*in module pytermor.color*), 39
`find_closest()` (*pytermor.color.Color16 class method*), 32

find_closest() (pytermor.color.Color256 class method), 34
 find_closest() (pytermor.color.ColorRGB class method), 37
 FixedString (class in pytermor.text), 59
 flip() (pytermor.style.Style method), 59
 format() (pytermor.utilnum.PrefixedUnitFormatter method), 62
 format() (pytermor.utilnum.TimeDeltaFormatter method), 63
 format_auto_float() (in module pytermor.utilnum), 64
 format_raw() (pytermor.utilnum.TimeDeltaFormatter method), 63
 format_si_binary() (in module pytermor.utilnum), 64
 format_si_metric() (in module pytermor.utilnum), 65
 format_thousand_sep() (in module pytermor.utilnum), 65
 format_time_delta() (in module pytermor.utilnum), 66
 format_value() (pytermor.color.Color16 method), 32
 format_value() (pytermor.color.Color256 method), 35
 format_value() (pytermor.color.ColorRGB method), 37
 FrozenText (class in pytermor.text), 59

G

GenericFilter (class in pytermor.utilstr), 67
 GenericPrinter (class in pytermor.utilstr), 68
 GenericStringPrinter (class in pytermor.utilstr), 68
 get_by_code() (pytermor.color.Color16 class method), 32
 get_by_code() (pytermor.color.Color256 class method), 35
 get_closing_seq() (in module pytermor.ansi), 29
 get_default() (pytermor.renderer.RendererManager class method), 54
 get_preferable_wrap_width() (in module pytermor.utilmisc), 61
 get_terminal_width() (in module pytermor.utilmisc), 61
 GOLD_1 (pytermor.cval.CVAL attribute), 45
 GOLD_2 (pytermor.cval.CVAL attribute), 45
 GOLD_3 (pytermor.cval.CVAL attribute), 45
 GRAY (pytermor.ansi.SeqIndex attribute), 25
 GRAY (pytermor.cval.CVAL attribute), 45
 GRAY_0 (pytermor.cval.CVAL attribute), 45
 GRAY_100 (pytermor.cval.CVAL attribute), 45
 GRAY_11 (pytermor.cval.CVAL attribute), 45
 GRAY_15 (pytermor.cval.CVAL attribute), 45
 GRAY_19 (pytermor.cval.CVAL attribute), 45
 GRAY_23 (pytermor.cval.CVAL attribute), 45
 GRAY_27 (pytermor.cval.CVAL attribute), 45
 GRAY_3 (pytermor.cval.CVAL attribute), 45
 GRAY_30 (pytermor.cval.CVAL attribute), 45
 GRAY_35 (pytermor.cval.CVAL attribute), 46
 GRAY_37 (pytermor.cval.CVAL attribute), 46
 GRAY_39 (pytermor.cval.CVAL attribute), 46
 GRAY_42 (pytermor.cval.CVAL attribute), 46
 GRAY_46 (pytermor.cval.CVAL attribute), 46
 GRAY_50 (pytermor.cval.CVAL attribute), 46
 GRAY_53 (pytermor.cval.CVAL attribute), 46
 GRAY_54 (pytermor.cval.CVAL attribute), 46
 GRAY_58 (pytermor.cval.CVAL attribute), 46
 GRAY_62 (pytermor.cval.CVAL attribute), 46
 GRAY_63 (pytermor.cval.CVAL attribute), 46
 GRAY_66 (pytermor.cval.CVAL attribute), 46
 GRAY_69 (pytermor.cval.CVAL attribute), 46
 GRAY_7 (pytermor.cval.CVAL attribute), 46
 GRAY_70 (pytermor.cval.CVAL attribute), 46
 GRAY_74 (pytermor.cval.CVAL attribute), 46
 GRAY_78 (pytermor.cval.CVAL attribute), 46
 GRAY_82 (pytermor.cval.CVAL attribute), 46
 GRAY_84 (pytermor.cval.CVAL attribute), 46
 GRAY_85 (pytermor.cval.CVAL attribute), 46
 GRAY_89 (pytermor.cval.CVAL attribute), 46
 GRAY_93 (pytermor.cval.CVAL attribute), 46
 GREEN (pytermor.ansi.SeqIndex attribute), 25
 GREEN (pytermor.cval.CVAL attribute), 46
 GREEN_2 (pytermor.cval.CVAL attribute), 46
 GREEN_3 (pytermor.cval.CVAL attribute), 46
 GREEN_4 (pytermor.cval.CVAL attribute), 46
 GREEN_5 (pytermor.cval.CVAL attribute), 46
 GREEN_YELLOW (pytermor.cval.CVAL attribute), 46

H

hex_to_hsv() (in module pytermor.color), 39
 hex_to_rgb() (in module pytermor.color), 39
 hex_value (pytermor.color.Color16 property), 34
 hex_value (pytermor.color.Color256 property), 36
 hex_value (pytermor.color.ColorRGB property), 38
 HI_BLUE (pytermor.ansi.SeqIndex attribute), 25
 HI_BLUE (pytermor.cval.CVAL attribute), 46
 HI_CYAN (pytermor.ansi.SeqIndex attribute), 25
 HI_CYAN (pytermor.cval.CVAL attribute), 46
 HI_GREEN (pytermor.ansi.SeqIndex attribute), 25
 HI_GREEN (pytermor.cval.CVAL attribute), 46
 HI_MAGENTA (pytermor.ansi.SeqIndex attribute), 25
 HI_MAGENTA (pytermor.cval.CVAL attribute), 47
 HI_RED (pytermor.ansi.SeqIndex attribute), 26
 HI_RED (pytermor.cval.CVAL attribute), 47
 HI_WHITE (pytermor.ansi.SeqIndex attribute), 26
 HI_WHITE (pytermor.cval.CVAL attribute), 47
 HI_YELLOW (pytermor.ansi.SeqIndex attribute), 26
 HI_YELLOW (pytermor.cval.CVAL attribute), 47
 HIDDEN (pytermor.ansi.SeqIndex attribute), 25
 HIDDEN_OFF (pytermor.ansi.SeqIndex attribute), 25
 HONEYDEW_2 (pytermor.cval.CVAL attribute), 47

HOT_PINK (*pytermor.cval.CVAL attribute*), 47
 HOT_PINK_2 (*pytermor.cval.CVAL attribute*), 47
 HOT_PINK_3 (*pytermor.cval.CVAL attribute*), 47
 HOT_PINK_4 (*pytermor.cval.CVAL attribute*), 47
 HOT_PINK_5 (*pytermor.cval.CVAL attribute*), 47
 hsv_to_hex() (*in module pytermor.color*), 40
 hsv_to_rgb() (*in module pytermor.color*), 40
 HtmlRenderer (*class in pytermor.renderer*), 52
 HYPERLINK (*pytermor.ansi.SeqIndex attribute*), 26

I

INDIAN_RED_1 (*pytermor.cval.CVAL attribute*), 47
 INDIAN_RED_2 (*pytermor.cval.CVAL attribute*), 47
 INDIAN_RED_3 (*pytermor.cval.CVAL attribute*), 47
 INDIAN_RED_4 (*pytermor.cval.CVAL attribute*), 47
 IntCode (*class in pytermor.ansi*), 23
 INVERSED (*pytermor.ansi.SeqIndex attribute*), 26
 INVERSED_OFF (*pytermor.ansi.SeqIndex attribute*), 26
 is_caching_allowed (*pytermor.renderer.AbstractRenderer property*), 52
 is_caching_allowed (*pytermor.renderer.HtmlRenderer property*), 53
 is_caching_allowed (*pytermor.renderer.NoOpRenderer property*), 53
 is_caching_allowed (*pytermor.renderer.SgrRenderer property*), 56
 is_caching_allowed (*pytermor.renderer.SgrRendererDebugger property*), 57
 is_caching_allowed (*pytermor.renderer.TmuxRenderer property*), 57
 is_format_allowed (*pytermor.renderer.AbstractRenderer property*), 52
 is_format_allowed (*pytermor.renderer.HtmlRenderer property*), 53
 is_format_allowed (*pytermor.renderer.NoOpRenderer property*), 53
 is_format_allowed (*pytermor.renderer.SgrRenderer property*), 56
 is_format_allowed (*pytermor.renderer.SgrRendererDebugger property*), 57
 is_format_allowed (*pytermor.renderer.TmuxRenderer property*), 57
 ITALIC (*pytermor.ansi.SeqIndex attribute*), 26
 ITALIC_OFF (*pytermor.ansi.SeqIndex attribute*), 26

K

KHAKI_1 (*pytermor.cval.CVAL attribute*), 47
 KHAKI_3 (*pytermor.cval.CVAL attribute*), 47

L

LIGHT_CORAL (*pytermor.cval.CVAL attribute*), 47
 LIGHT_CYAN_1 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_CYAN_3 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GOLDENROD_1 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GOLDENROD_2 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GOLDENROD_3 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GOLDENROD_4 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GOLDENROD_5 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GREEN (*pytermor.cval.CVAL attribute*), 47
 LIGHT_GREEN_2 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_PINK_1 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_PINK_2 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_PINK_3 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_SALMON_1 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_SALMON_2 (*pytermor.cval.CVAL attribute*), 47
 LIGHT_SALMON_3 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_SEA_GREEN (*pytermor.cval.CVAL attribute*), 48
 LIGHT_SKY_BLUE_1 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_SKY_BLUE_2 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_SKY_BLUE_3 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_SLATE_BLUE (*pytermor.cval.CVAL attribute*), 48
 LIGHT_SLATE_GRAY (*pytermor.cval.CVAL attribute*), 48
 LIGHT_STEEL_BLUE_1 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_STEEL_BLUE_2 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_STEEL_BLUE_3 (*pytermor.cval.CVAL attribute*), 48
 LIGHT_YELLOW_3 (*pytermor.cval.CVAL attribute*), 48
 ljust_sgr() (*in module pytermor.utilstr*), 72
 LogicError, 42

M

MAGENTA (*pytermor.ansi.SeqIndex attribute*), 26
 MAGENTA (*pytermor.cval.CVAL attribute*), 48
 MAGENTA_1 (*pytermor.cval.CVAL attribute*), 48
 MAGENTA_2 (*pytermor.cval.CVAL attribute*), 48
 MAGENTA_3 (*pytermor.cval.CVAL attribute*), 48
 MAGENTA_4 (*pytermor.cval.CVAL attribute*), 48
 MAGENTA_5 (*pytermor.cval.CVAL attribute*), 48
 MAGENTA_6 (*pytermor.cval.CVAL attribute*), 48
 make_color_256() (*in module pytermor.ansi*), 30
 make_color_rgb() (*in module pytermor.ansi*), 30
 make_erase_in_line() (*in module pytermor.ansi*), 30
 make_hyperlink_part() (*in module pytermor.ansi*), 30
 make_set_cursor_x_abs() (*in module pytermor.ansi*), 31
 max_len (*pytermor.utilnum.PrefixedUnitFormatter property*), 62
 max_len (*pytermor.utilnum.TimeDeltaFormatter property*), 63
 MEDIUM_ORCHID_1 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_ORCHID_2 (*pytermor.cval.CVAL attribute*), 48

MEDIUM_ORCHID_3 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_ORCHID_4 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_1 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_2 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_3 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_4 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_5 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_6 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_PURPLE_7 (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_SPRING_GREEN (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_TURQUOISE (*pytermor.cval.CVAL attribute*), 48
 MEDIUM_VIOLET_RED (*pytermor.cval.CVAL attribute*), 49
 MISTY_ROSE_1 (*pytermor.cval.CVAL attribute*), 49
 MISTY_ROSE_3 (*pytermor.cval.CVAL attribute*), 49
 module
 pytermor.ansi, 23
 pytermor.color, 31
 pytermor.common, 42
 pytermor.cval, 43
 pytermor.renderer, 51
 pytermor.style, 58
 pytermor.text, 59
 pytermor.utilmisc, 60
 pytermor.utilnum, 62
 pytermor.utilstr, 67

N

name (*pytermor.color.Color16 property*), 34
 name (*pytermor.color.Color256 property*), 36
 name (*pytermor.color.ColorRGB property*), 38
 NAVAJO_WHITE_1 (*pytermor.cval.CVAL attribute*), 49
 NAVAJO_WHITE_3 (*pytermor.cval.CVAL attribute*), 49
 NAVY_BLUE (*pytermor.cval.CVAL attribute*), 49
 NO_ANSI (*pytermor.renderer.OutputMode attribute*), 54
 NonPrintablesOmniVisualizer (*class in pytermor.utilstr*), 68
 NonPrintablesStringVisualizer (*class in pytermor.utilstr*), 68
 NOOP_COLOR (*in module pytermor.color*), 42
 NOOP_SEQ (*in module pytermor.ansi*), 31
 NOOP_STYLE (*in module pytermor.style*), 59
 NoOpRenderer (*class in pytermor.renderer*), 53

O

OmniMapper (*class in pytermor.utilstr*), 68
 OmniSanitizer (*class in pytermor.utilstr*), 69
 ORANGE_1 (*pytermor.cval.CVAL attribute*), 49
 ORANGE_2 (*pytermor.cval.CVAL attribute*), 49
 ORANGE_3 (*pytermor.cval.CVAL attribute*), 49
 ORANGE_4 (*pytermor.cval.CVAL attribute*), 49
 ORANGE_RED_1 (*pytermor.cval.CVAL attribute*), 49
 ORCHID_1 (*pytermor.cval.CVAL attribute*), 49
 ORCHID_2 (*pytermor.cval.CVAL attribute*), 49

ORCHID_3 (*pytermor.cval.CVAL attribute*), 49
 OutputMode (*class in pytermor.renderer*), 54
 OVERLINED (*pytermor.ansi.SeqIndex attribute*), 26
 OVERLINED_OFF (*pytermor.ansi.SeqIndex attribute*), 26

P

PALE_GREEN_1 (*pytermor.cval.CVAL attribute*), 49
 PALE_GREEN_2 (*pytermor.cval.CVAL attribute*), 49
 PALE_GREEN_3 (*pytermor.cval.CVAL attribute*), 49
 PALE_GREEN_4 (*pytermor.cval.CVAL attribute*), 49
 PALE_TURQUOISE_1 (*pytermor.cval.CVAL attribute*), 49
 PALE_TURQUOISE_4 (*pytermor.cval.CVAL attribute*), 49
 PALE_VIOLET_RED_1 (*pytermor.cval.CVAL attribute*), 49
 params (*pytermor.ansi.Sequence property*), 27
 params (*pytermor.ansi.SequenceCSI property*), 27
 params (*pytermor.ansi.SequenceFe property*), 27
 params (*pytermor.ansi.SequenceOSC property*), 28
 params (*pytermor.ansi.SequenceSGR property*), 28
 params (*pytermor.ansi.SequenceST property*), 29
 PINK_1 (*pytermor.cval.CVAL attribute*), 49
 PINK_3 (*pytermor.cval.CVAL attribute*), 49
 PLUM_1 (*pytermor.cval.CVAL attribute*), 49
 PLUM_2 (*pytermor.cval.CVAL attribute*), 49
 PLUM_3 (*pytermor.cval.CVAL attribute*), 49
 PLUM_4 (*pytermor.cval.CVAL attribute*), 49
 PREFIX_ZERO_SI (*in module pytermor.utilnum*), 66
 PrefixedUnitFormatter (*class in pytermor.utilnum*), 62
 PREFIXES_SI (*in module pytermor.utilnum*), 66
 PrinterExtra (*class in pytermor.utilstr*), 69
 PURPLE (*pytermor.cval.CVAL attribute*), 49
 PURPLE_2 (*pytermor.cval.CVAL attribute*), 49
 PURPLE_3 (*pytermor.cval.CVAL attribute*), 49
 PURPLE_4 (*pytermor.cval.CVAL attribute*), 49
 PURPLE_6 (*pytermor.cval.CVAL attribute*), 50
 pytermor.ansi
 module, 23
 pytermor.color
 module, 31
 pytermor.common
 module, 42
 pytermor.cval
 module, 43
 pytermor.renderer
 module, 51
 pytermor.style
 module, 58
 pytermor.text
 module, 59
 pytermor.utilmisc
 module, 60
 pytermor.utilnum
 module, 62
 pytermor.utilstr

module, 67

R

RED (*pytermor.ansi.SeqIndex* attribute), 26
 RED (*pytermor.cval.CVAL* attribute), 50
 RED_1 (*pytermor.cval.CVAL* attribute), 50
 RED_3 (*pytermor.cval.CVAL* attribute), 50
 RED_4 (*pytermor.cval.CVAL* attribute), 50
 render() (in module *pytermor.text*), 60
 render() (*pytermor.renderer.AbstractRenderer* method), 52
 render() (*pytermor.renderer.HtmlRenderer* method), 53
 render() (*pytermor.renderer.NoOpRenderer* method), 53
 render() (*pytermor.renderer.SgrRenderer* method), 55
 render() (*pytermor.renderer.SgrRendererDebugger* method), 56
 render() (*pytermor.renderer.TmuxRenderer* method), 57
 Renderable (class in *pytermor.text*), 59
 RendererManager (class in *pytermor.renderer*), 54
 RESET (*pytermor.ansi.SeqIndex* attribute), 26
 resolve() (in module *pytermor.color*), 40
 resolve() (*pytermor.ansi.IntCode* class method), 23
 resolve() (*pytermor.color.Color16* class method), 33
 resolve() (*pytermor.color.Color256* class method), 35
 resolve() (*pytermor.color.ColorRGB* class method), 37
 rgb_to_hex() (in module *pytermor.color*), 41
 rgb_to_hsv() (in module *pytermor.color*), 41
 rjust_sgr() (in module *pytermor.utilstr*), 72
 ROSY_BROWN (*pytermor.cval.CVAL* attribute), 50
 ROYAL_BLUE_1 (*pytermor.cval.CVAL* attribute), 50

S

SALMON_1 (*pytermor.cval.CVAL* attribute), 50
 SANDY_BROWN (*pytermor.cval.CVAL* attribute), 50
 SEA_GREEN_1 (*pytermor.cval.CVAL* attribute), 50
 SEA_GREEN_2 (*pytermor.cval.CVAL* attribute), 50
 SEA_GREEN_3 (*pytermor.cval.CVAL* attribute), 50
 SEA_GREEN_4 (*pytermor.cval.CVAL* attribute), 50
 SeqIndex (class in *pytermor.ansi*), 24
 Sequence (class in *pytermor.ansi*), 26
 SequenceCSI (class in *pytermor.ansi*), 27
 SequenceFe (class in *pytermor.ansi*), 27
 SequenceOSC (class in *pytermor.ansi*), 27
 SequenceSGR (class in *pytermor.ansi*), 28
 SequenceST (class in *pytermor.ansi*), 29
 set_default() (*pytermor.renderer.RendererManager* class method), 54
 set_default_format_always() (*pytermor.renderer.RendererManager* class method), 54
 set_default_format_never() (*pytermor.renderer.RendererManager* class method), 55

set_format_always() (*pytermor.renderer.SgrRendererDebugger* method), 57
 set_format_auto() (*pytermor.renderer.SgrRendererDebugger* method), 57
 set_format_never() (*pytermor.renderer.SgrRendererDebugger* method), 57
 SgrRenderer (class in *pytermor.renderer*), 55
 SgrRendererDebugger (class in *pytermor.renderer*), 56
 SgrStringReplacer (class in *pytermor.utilstr*), 69
 SKY_BLUE_1 (*pytermor.cval.CVAL* attribute), 50
 SKY_BLUE_2 (*pytermor.cval.CVAL* attribute), 50
 SKY_BLUE_3 (*pytermor.cval.CVAL* attribute), 50
 SLATE_BLUE_1 (*pytermor.cval.CVAL* attribute), 50
 SLATE_BLUE_2 (*pytermor.cval.CVAL* attribute), 50
 SLATE_BLUE_3 (*pytermor.cval.CVAL* attribute), 50
 SPRING_GREEN_1 (*pytermor.cval.CVAL* attribute), 50
 SPRING_GREEN_2 (*pytermor.cval.CVAL* attribute), 50
 SPRING_GREEN_3 (*pytermor.cval.CVAL* attribute), 50
 SPRING_GREEN_4 (*pytermor.cval.CVAL* attribute), 50
 SPRING_GREEN_5 (*pytermor.cval.CVAL* attribute), 50
 SPRING_GREEN_6 (*pytermor.cval.CVAL* attribute), 50
 STEEL_BLUE (*pytermor.cval.CVAL* attribute), 50
 STEEL_BLUE_1 (*pytermor.cval.CVAL* attribute), 50
 STEEL_BLUE_2 (*pytermor.cval.CVAL* attribute), 50
 STEEL_BLUE_3 (*pytermor.cval.CVAL* attribute), 50
 String (class in *pytermor.text*), 59
 StringHexPrinter (class in *pytermor.utilstr*), 69
 StringMapper (class in *pytermor.utilstr*), 70
 StringReplacer (class in *pytermor.utilstr*), 70
 StringUcpPrinter (class in *pytermor.utilstr*), 70
 StrType (in module *pytermor.common*), 42
 Style (class in *pytermor.style*), 58
 Styles (class in *pytermor.style*), 59

T

T (in module *pytermor.common*), 42
 TAN (*pytermor.cval.CVAL* attribute), 50
 Text (class in *pytermor.text*), 59
 THISTLE_1 (*pytermor.cval.CVAL* attribute), 50
 THISTLE_3 (*pytermor.cval.CVAL* attribute), 51
 TimeDeltaFormatter (class in *pytermor.utilnum*), 62
 TimeUnit (class in *pytermor.utilnum*), 63
 TmuxRenderer (class in *pytermor.renderer*), 57
 to_hsv() (*pytermor.color.Color16* method), 33
 to_hsv() (*pytermor.color.Color256* method), 35
 to_hsv() (*pytermor.color.ColorRGB* method), 37
 to_rgb() (*pytermor.color.Color16* method), 33
 to_rgb() (*pytermor.color.Color256* method), 35
 to_rgb() (*pytermor.color.ColorRGB* method), 37
 to_sgr() (*pytermor.color.Color16* method), 33
 to_sgr() (*pytermor.color.Color256* method), 35

[to_sgr\(\)](#) (*pytermor.color.ColorRGB method*), 37
[to_tmx\(\)](#) (*pytermor.color.Color16 method*), 33
[to_tmx\(\)](#) (*pytermor.color.Color256 method*), 36
[to_tmx\(\)](#) (*pytermor.color.ColorRGB method*), 38
[total_size\(\)](#) (*in module pytermor.utilmisc*), 61
[TRUE_COLOR](#) (*pytermor.renderer.OutputMode attribute*),
[54](#)
[TURQUOISE_2](#) (*pytermor.cval.CVAL attribute*), 51
[TURQUOISE_4](#) (*pytermor.cval.CVAL attribute*), 51

U

[UNDERLINED](#) (*pytermor.ansi.SeqIndex attribute*), 26
[UNDERLINED_OFF](#) (*pytermor.ansi.SeqIndex attribute*), 26
[UnderlinedCurlySequenceSGR](#) (*class in pytermor.ansi*), 29
[UserAbort](#), 42
[UserCancel](#), 42

V

[variations](#) (*pytermor.color.ColorRGB property*), 38
[VIOLET](#) (*pytermor.cval.CVAL attribute*), 51

W

[wait_key\(\)](#) (*in module pytermor.utilmisc*), 61
[WHEAT_1](#) (*pytermor.cval.CVAL attribute*), 51
[WHEAT_4](#) (*pytermor.cval.CVAL attribute*), 51
[WHITE](#) (*pytermor.ansi.SeqIndex attribute*), 26
[WHITE](#) (*pytermor.cval.CVAL attribute*), 51
[wrap_sgr\(\)](#) (*in module pytermor.utilstr*), 72

X

[XTERM_16](#) (*pytermor.renderer.OutputMode attribute*), 54
[XTERM_256](#) (*pytermor.renderer.OutputMode attribute*),
[54](#)

Y

[YELLOW](#) (*pytermor.ansi.SeqIndex attribute*), 26
[YELLOW](#) (*pytermor.cval.CVAL attribute*), 51
[YELLOW_1](#) (*pytermor.cval.CVAL attribute*), 51
[YELLOW_2](#) (*pytermor.cval.CVAL attribute*), 51
[YELLOW_3](#) (*pytermor.cval.CVAL attribute*), 51
[YELLOW_4](#) (*pytermor.cval.CVAL attribute*), 51
[YELLOW_5](#) (*pytermor.cval.CVAL attribute*), 51
[YELLOW_6](#) (*pytermor.cval.CVAL attribute*), 51