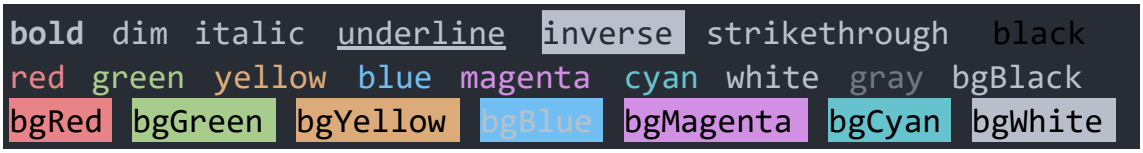
Chalk

> *Terminal string styling done right*



Highlights

- Expressive API
- Highly performant
- Ability to nest styles
- 256/Truecolor color support
- Auto-detects color support
- Doesn't extend `String.prototype`
- Clean and focused
- Actively maintained
- Used by ~50,000 packages as of January 1, 2020

Install

```
$ npm install chalk
```

Usage

```
const chalk = require('chalk');

console.log(chalk.blue('Hello world!'));
```

Chalk comes with an easy to use composable API where you just chain and nest the styles you want.

```
const chalk = require('chalk');
const log = console.log;

// Combine styled and normal strings
log(chalk.blue('Hello') + ' World' + chalk.red('!'));

// Compose multiple styles using the chainable API
log(chalk.blue.bgRed.bold('Hello world!'));

// Pass in multiple arguments
log(chalk.blue('Hello', 'World!', 'Foo', 'bar', 'biz', 'baz'));

// Nest styles
log(chalk.red('Hello', chalk.underline.bgBlue('world') + '!'));

// Nest styles of the same type even (color, underline, background)
```

```
log(chalk.green(
    'I am a green line ' +
    chalk.blue.underline.bold('with a blue substring') +
    ' that becomes green again!'
));

// ES2015 template literal
log(`
CPU: ${chalk.red('90%')}
RAM: ${chalk.green('40%')}
DISK: ${chalk.yellow('70%')}
`);

// ES2015 tagged template literal
log(chalk`
CPU: {red ${cpu.totalPercent}%}
RAM: {green ${ram.used / ram.total * 100}%}
DISK: {rgb(255,131,0) ${disk.used / disk.total * 100}%}
`);

// Use RGB colors in terminal emulators that support it.
log(chalk.keyword('orange')('Yay for orange colored text!'));
log(chalk.rgb(123, 45, 67).underline('Underlined reddish color'));
log(chalk.hex('#DEADED').bold('Bold gray!'));
```

Easily define your own themes:

```
const chalk = require('chalk');

const error = chalk.bold.red;
const warning = chalk.keyword('orange');

console.log(error('Error!'));
console.log(warning('Warning!'));
```

Take advantage of console.log string substitution:

```
const name = 'Sindre';
console.log(chalk.green('Hello %s'), name);
//=> 'Hello Sindre'
```

API

chalk.`<style>[.<style>...](string, [string...])`

Example: `chalk.red.bold.underline('Hello', 'world');`

Chain styles and call the last one as a method with a string argument. Order doesn't matter, and later styles take precedent in case of a conflict. This simply means that `chalk.red.yellow.green` is equivalent to `chalk.green`.

Multiple arguments will be separated by space.

chalk.level

Specifies the level of color support.

Color support is automatically detected, but you can override it by setting the `level` property. You should however only do this in your own code as it applies globally to all Chalk consumers.

If you need to change this in a reusable module, create a new instance:

```
const ctx = new chalk.Instance({level: 0});
```

Level	Description
0	All colors disabled
1	Basic color support (16 colors)
2	256 color support
3	Truecolor support (16 million colors)

chalk.supportsColor

Detect whether the terminal supports color. Used internally and handled for you, but exposed for convenience.

Can be overridden by the user with the flags `--color` and `--no-color`. For situations where using `--color` is not possible, use the environment variable `FORCE_COLOR=1` (level 1), `FORCE_COLOR=2` (level 2), or `FORCE_COLOR=3` (level 3) to forcefully enable color, or `FORCE_COLOR=0` to forcefully disable. The use of `FORCE_COLOR` overrides all other color support checks.

Explicit 256/Truecolor mode can be enabled using the `--color=256` and `--color=16m` flags, respectively.

chalk.stderr and chalk.stderr.supportsColor

`chalk.stderr` contains a separate instance configured with color support detected for `stderr` stream instead of `stdout`. Override rules from `chalk.supportsColor` apply to this too. `chalk.stderr.supportsColor` is exposed for convenience.

Styles

Modifiers

- `reset` - Resets the current color chain.
- `bold` - Make text bold.
- `dim` - Emitting only a small amount of light.
- `italic` - Make text italic. *(Not widely supported)*
- `underline` - Make text underline. *(Not widely supported)*
- `inverse` - Inverse background and foreground colors.
- `hidden` - Prints the text, but makes it invisible.
- `strikethrough` - Puts a horizontal line through the center of the text. *(Not widely supported)*

- `visible` - Prints the text only when Chalk has a color level > 0. Can be useful for things that are purely cosmetic.

Colors

- `black`
- `red`
- `green`
- `yellow`
- `blue`
- `magenta`
- `cyan`
- `white`
- `blackBright` (alias: `gray`, `grey`)
- `redBright`
- `greenBright`
- `yellowBright`
- `blueBright`
- `magentaBright`
- `cyanBright`
- `whiteBright`

Background colors

- `bgBlack`
- `bgRed`
- `bgGreen`
- `bgYellow`
- `bgBlue`
- `bgMagenta`
- `bgCyan`
- `bgWhite`
- `bgBlackBright` (alias: `bgGray`, `bgGrey`)
- `bgRedBright`
- `bgGreenBright`
- `bgYellowBright`
- `bgBlueBright`
- `bgMagentaBright`
- `bgCyanBright`
- `bgWhiteBright`

Tagged template literal

Chalk can be used as a tagged template literal.

```
const chalk = require('chalk');

const miles = 18;
const calculateFeet = miles => miles * 5280;
```

```
console.log(chalk`
    There are {bold 5280 feet} in a mile.
    In {bold ${miles} miles}, there are {green.bold ${calculateFeet(miles)} feet}.
`);
```

Blocks are delimited by an opening curly brace (`{`), a style, some content, and a closing curly brace (`}`).

Template styles are chained exactly like normal Chalk styles. The following three statements are equivalent:

```
console.log(chalk.bold.rgb(10, 100, 200)('Hello!'));
console.log(chalk.bold.rgb(10, 100, 200)`Hello!`);
console.log(chalk`{bold.rgb(10,100,200) Hello!}`);
```

Note that function styles (`rgb()` , `hsl()` , `keyword()` , etc.) may not contain spaces between parameters.

All interpolated values (`chalk`${foo}``) are converted to strings via the `.toString()` method. All curly braces (`{` and `}`) in interpolated value strings are escaped.

256 and Truecolor color support

Chalk supports 256 colors and Truecolor (16 million colors) on supported terminal apps.

Colors are downsampled from 16 million RGB values to an ANSI color format that is supported by the terminal emulator (or by specifying `{level: n}` as a Chalk option). For example, Chalk configured to run at level 1 (basic color support) will downsample an RGB value of #FF0000 (red) to 31 (ANSI escape for red).

Examples:

- `chalk.hex('#DEADED').underline('Hello, world!')`
- `chalk.keyword('orange')('Some orange text')`
- `chalk.rgb(15, 100, 204).inverse('Hello!')`

Background versions of these models are prefixed with `bg` and the first level of the module capitalized (e.g. `keyword` for foreground colors and `bgKeyword` for background colors).

- `chalk.bgHex('#DEADED').underline('Hello, world!')`
- `chalk.bgKeyword('orange')('Some orange text')`
- `chalk.bgRgb(15, 100, 204).inverse('Hello!')`

The following color models can be used:

- `rgb` - Example: `chalk.rgb(255, 136, 0).bold('Orange!')`
- `hex` - Example: `chalk.hex('#FF8800').bold('Orange!')`
- `keyword` (CSS keywords) - Example: `chalk.keyword('orange').bold('Orange!')`
- `hsl` - Example: `chalk.hsl(32, 100, 50).bold('Orange!')`
- `hsv` - Example: `chalk.hsv(32, 100, 100).bold('Orange!')`
- `hwb` - Example: `chalk.hwb(32, 0, 50).bold('Orange!')`
- `ansi` - Example: `chalk.ansi(31).bgAnsi(93)('red on yellowBright')`
- `ansi256` - Example: `chalk.bgAnsi256(194)('Honeydew, more or less')`

Windows

If you're on Windows, do yourself a favor and use Windows Terminal instead of `cmd.exe`.

Origin story

colors.js used to be the most popular string styling module, but it has serious deficiencies like extending `String.prototype` which causes all kinds of problems and the package is unmaintained. Although there are other packages, they either do too much or not enough. Chalk is a clean and focused alternative.

chalk for enterprise

Available as part of the Tidelift Subscription.

The maintainers of chalk and thousands of other packages are working with Tidelift to deliver commercial support and maintenance for the open source dependencies you use to build your applications. Save time, reduce risk, and improve code health, while paying the maintainers of the exact dependencies you use. Learn more.

Related

- chalk-cli - CLI for this module
- ansi-styles - ANSI escape codes for styling strings in the terminal
- supports-color - Detect whether a terminal supports color
- strip-ansi - Strip ANSI escape codes
- strip-ansi-stream - Strip ANSI escape codes from a stream
- has-ansi - Check if a string has ANSI escape codes
- ansi-regex - Regular expression for matching ANSI escape codes
- wrap-ansi - Wordwrap a string with ANSI escape codes
- slice-ansi - Slice a string with ANSI escape codes
- color-convert - Converts colors between different models
- chalk-animation - Animate strings in the terminal
- gradient-string - Apply color gradients to strings
- chalk-pipe - Create chalk style schemes with simpler style strings
- terminal-link - Create clickable links in the terminal

Maintainers

- Sindre Sorhus
- Josh Junon