

`ll.ansistyle` is a module that helps colorize terminal output with ANSI escape sequences.

Color values are integers between -1 and 511 (octal 0777). The lowest 3 bits are used for the background colors 0-7. Bits 3-5 are used for the foreground color. Bit 6 (0100) is used for bold (or other shades of the color 0-7, depending on your terminal). Bit 7 (0200) is used for underlined text and bit 8 (0400) is used for blinking text. The color value -1 means “don't change the color”.

To add color switching escape sequences to normal output test, use a `Colorizer` object like this:

```
>>> from ll import ansistyle
>>> c = ansistyle.Colorizer()
>>> list(c.feed("spam", 0157, "eggs", None))
['spam', '\x1b[1;35;47m', 'eggs', '\x1b[0m']
```

For more info see the description of the `feed`.

There is a second level API for `ansistyle.Colorizer`, which can be used like this:

```
>>> from ll import ansistyle
>>> list(ansistyle.Text("spam", 0157, "eggs").parts())
['spam', '\x1b[1;35;47m', 'eggs', '\x1b[0m']
```

Furthermore you can print `ansistyle.Text` instances directly to get colored output.

For more information see the documentation for `ansistyle.Text`.

1. class `Colorizer(object)`:

A `Colorizer` object manages the current color and style and will intersperse normal output text with ANSI escape sequences for switching output color and style.

1.1. def `__init__(self, colored=True)`:

Create a `Colorizer` instance. If `colored` is false, output will never contain any color/style switching escape sequences.

1.2. def `pushcolor(self, color)`:

Push `color` onto the color stack

1.3. def `popcolor(self)`:

Pop a color from the color stack.

1.4. def `_switchcolor(self, color)`:

1.5. def `feed(self, *strings)`:

This method is a generator and will yield all the strings in `strings` with interspersed color switching escape sequences. Items in `strings` can be the following:

Strings (str and unicode)

Strings will be output by `feed` in the appropriate spot.

Numbers

A number in the argument sequence will switch to that color value.

None

This will switch back to the default color (This is different from using the color number 0070, because 0070 will only switch colors if there is some output string afterwards).

Sequences

Those will be recursively fed to `feed` with the following added functionality: The color that was active before the start of the sequence will be restored after the end.

2. class `Text(list)`:

A colored string. A `Text` object is a sequence, the sequence items may either be strings or `Text` objects themselves.

2.1. `def __init__(self, *content):`

2.2. `def append(self, *others):`

2.3. `def __call__(self, *others):`

2.4. `def insert(self, index, *others):`

2.5. `def __repr__(self):`

2.6. `def parts(self, colored=True):`

This generator yields the strings that will make up the final colored string.

2.7. `def string(self, colored=True):`

Return the resulting string (with escape sequences, if `colored` is true).

2.8. `def __str__(self):`

Return the resulting string with ANSI color escape sequences.

3. class `EscapedText(Text)`:

An extension to the `Text` class. Special characters can be replaced by customized `Text` objects via the `escapechar` method.

3.1. `def __init__(self, *content):`

3.2. `def escapechar(self, char):`

Return a replacement `Text` object for the character `char` or `char` itself, if the character should be used as is. This method should be overwritten by subclasses.