term-image

AnonymouX47

Jan 09, 2023

CONTENTS:

Attention: Under Construction - There might be incompatible changes between minor versions of version zero!

If you want to use `term-image` in a project while it's still on version zero, ensure you pin the dependency to a specific minor version e.g `>=0.4,<0.5`.

CHAPTER

ONE

INSTALLATION

1.1 Requirements

- Operating System: Unix / Linux / MacOS X / Windows (limited support, see the *FAQs*)
- Python >= 3.7
- A terminal emulator with **any** of the following:
  - support for the Kitty graphics protocol.
  - support for the iTerm2 inline image protocol.
  - full Unicode support and ANSI 24-bit color support

  Plans to support a wider variety of terminal emulators are in motion (see *Planned Features*).

1.2 Steps

The latest **stable** version can be installed from PyPI using `pip`:

```
pip install term-image
```

The **development** version can be installed thus:

NOTE: it's recommended to install in an isolated virtual environment, can be created by any means.

Clone the repository,

```
git clone https://github.com/AnonymouX47/term-image.git
```

then navigate into the local repository

```
cd term-image
```

and run

```
pip install .
```

1.3 Supported Terminal Emulators

Some terminals emulators that have been tested to meet the requirements for at least one render style include:

- **libvte**-based terminal emulators such as:
  - Gnome Terminal
  - Terminator
  - Tilix
- Kitty
- Konsole
- iTerm2
- WezTerm
- Alacritty
- Windows Terminal
- MinTTY (on Windows)
- Termux (on Android)

For style-specific support, see the descriptions of the respective *Image Classes* or the **Render Styles** section towards the bottom of the command-line help text (i.e the output of `term-image --help`).

Note: If you've tested `term-image` on any other terminal emulator that meets all requirements, please mention the name in a new thread under this discussion.

Also, if you're having an issue with terminal support, you may report or check information about it in the discussion linked above.

Note: Some terminal emulators support 24-bit color escape sequences but have a 256-color pallete. This will limit color reproduction.

CHAPTER

TWO

GALLERY

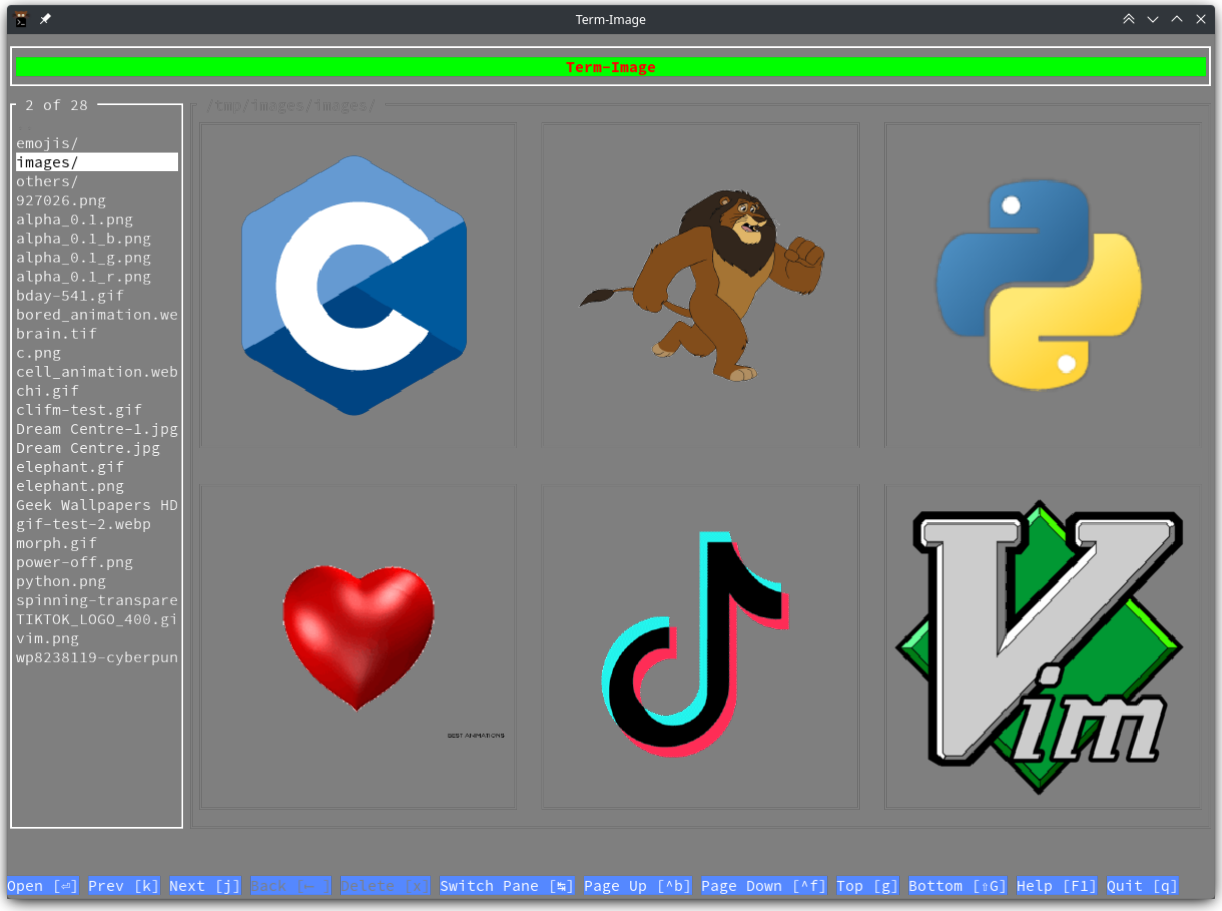


Fig. 1: **kitty** render style in Kitty

Fig. 2: **kitty** render style in Konsole

Fig. 3: **iterm2** render style in Konsole

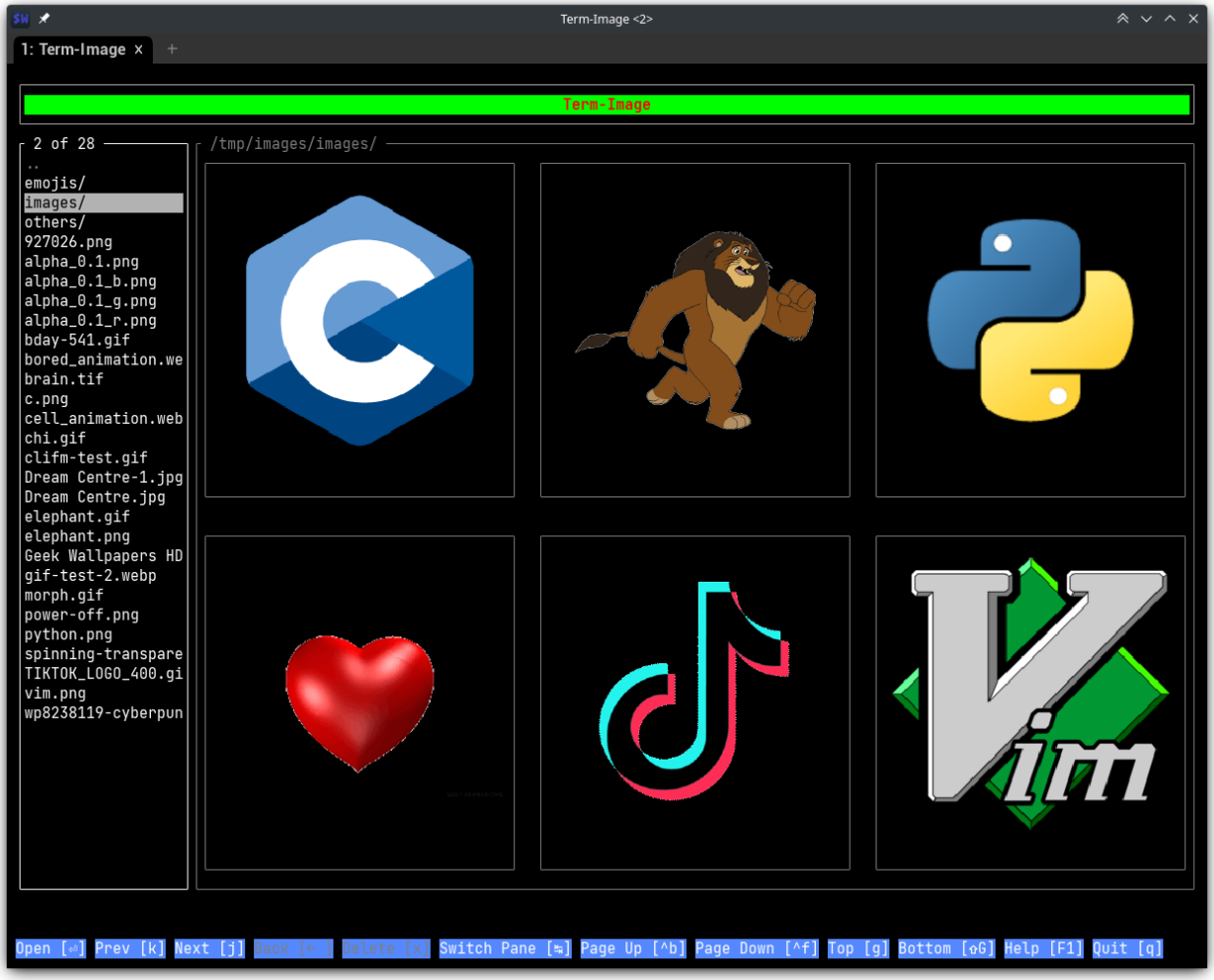


Fig. 4: **iterm2** render style in Wezterm

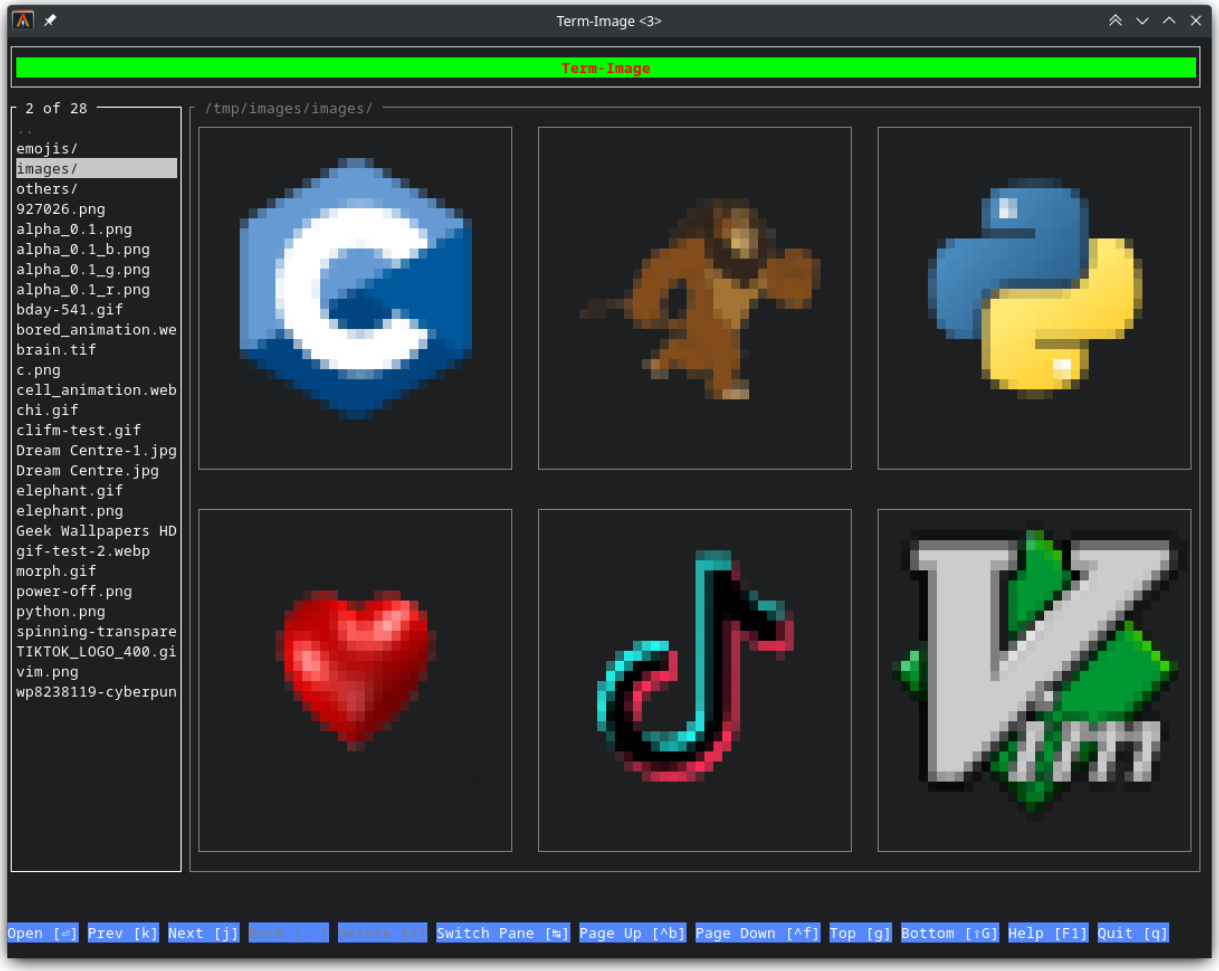


Fig. 5: **block** render style in Alacritty

Fig. 6: **block** render style in Terminator

Fig. 7: **block** render style in Termux

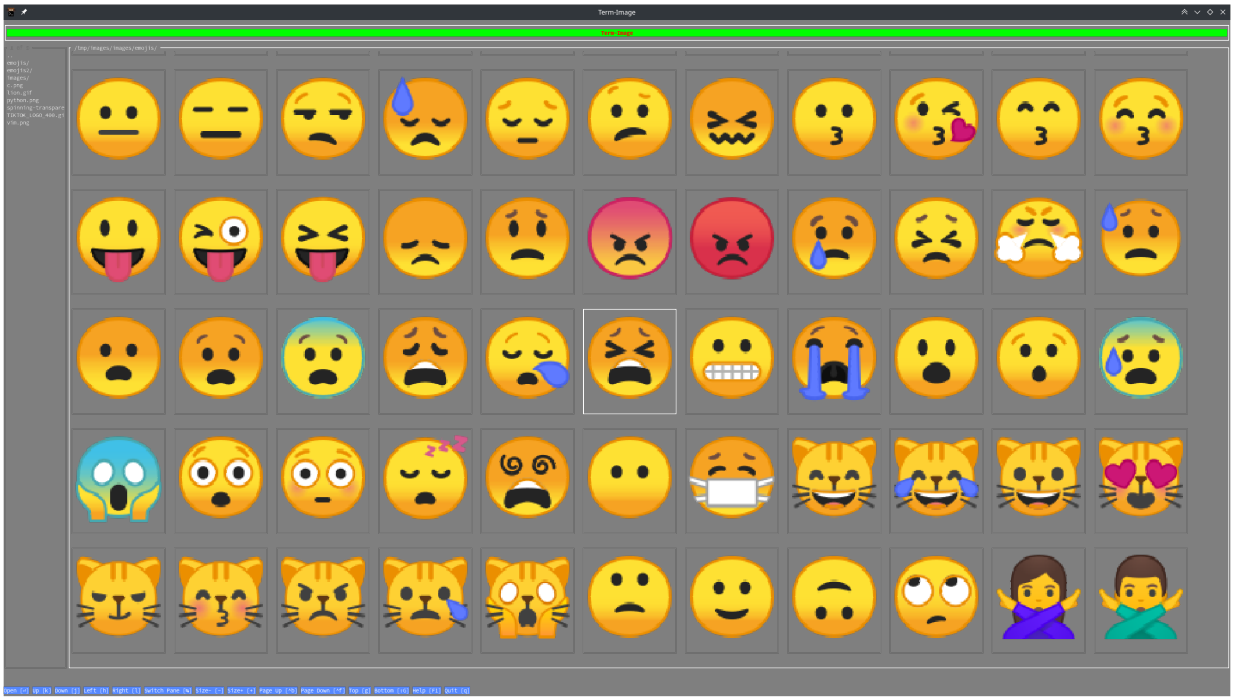


Fig. 8: Handles large image grids efficiently

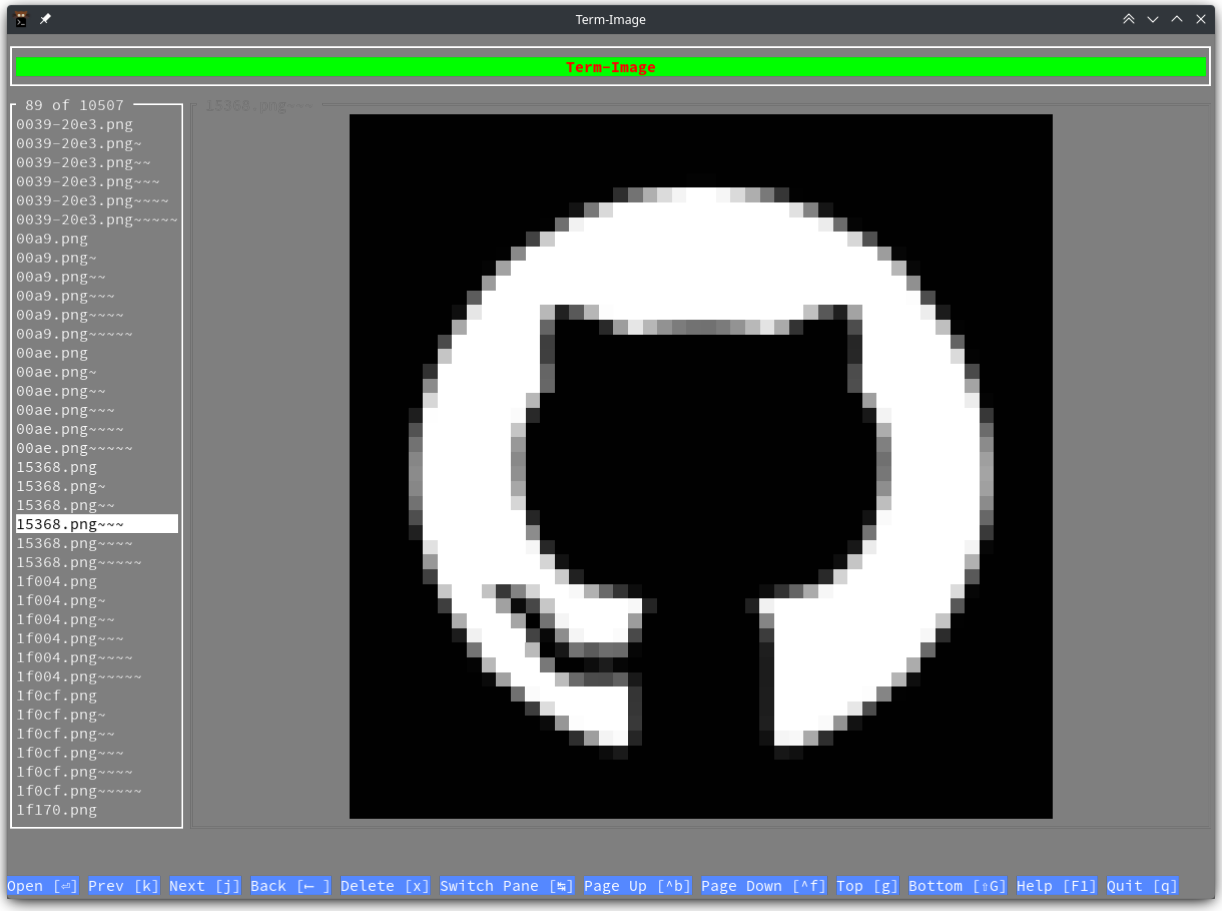


Fig. 9: Loads large directories efficiently

Fig. 10: Displays high resolution images efficiently

CHAPTER

THREE

LIBRARY DOCUMENTATION

Attention: Under Construction - There might be incompatible interface changes between minor versions of version zero!

If you want to use the library in a project while it's still on version zero, ensure you pin the dependency to a specific minor version e.g `>=0.4,<0.5`.

On this note, you probably also want to switch to the specific documentation for the version you're using (somewhere at the lower left corner of this page).

3.1 Tutorial

This is a basic introduction to using the library. Please refer to the *Reference* for detailed description of the features and functionality provided by the library.

For this tutorial we'll be using the image below:

The image has a resolution of **288x288 pixels**.

Note: All the samples in this tutorial occured in a terminal window of **255 columns by 70 lines**.

3.1.1 Creating an instance

Image instances can be created using the convinience functions *AutoImage()*, *from_file()* and *from_url()*. These automatically detect the best style supported by the *active terminal*.

Instances can also be created using the *Image Classes* directly via their respective constructors or *from_file()* and *from_url()* methods.

If the file is stored on your local filesystem:

```
from term_image.image import from_file

image = from_file("path/to/python.png")
```

You can also use a URL if you don't have the file stored locally:

```
from term_image.image import from_url

image = from_url("https://raw.githubusercontent.com/AnonymouX47/term-image/main/docs/
↪source/resources/tutorial/python.png")
```

The library can also be used with PIL image instances:

```
from PIL import Image
from term_image.image import AutoImage

img = Image.open("python.png")
image = AutoImage(img)
```

3.1.2 Rendering an image

Rendering an image is simply the process of converting it (per-frame for *animated* images) into text (a string).

Hint: To display the rendered image in the following steps, just pass the string as an argument to `print()`.

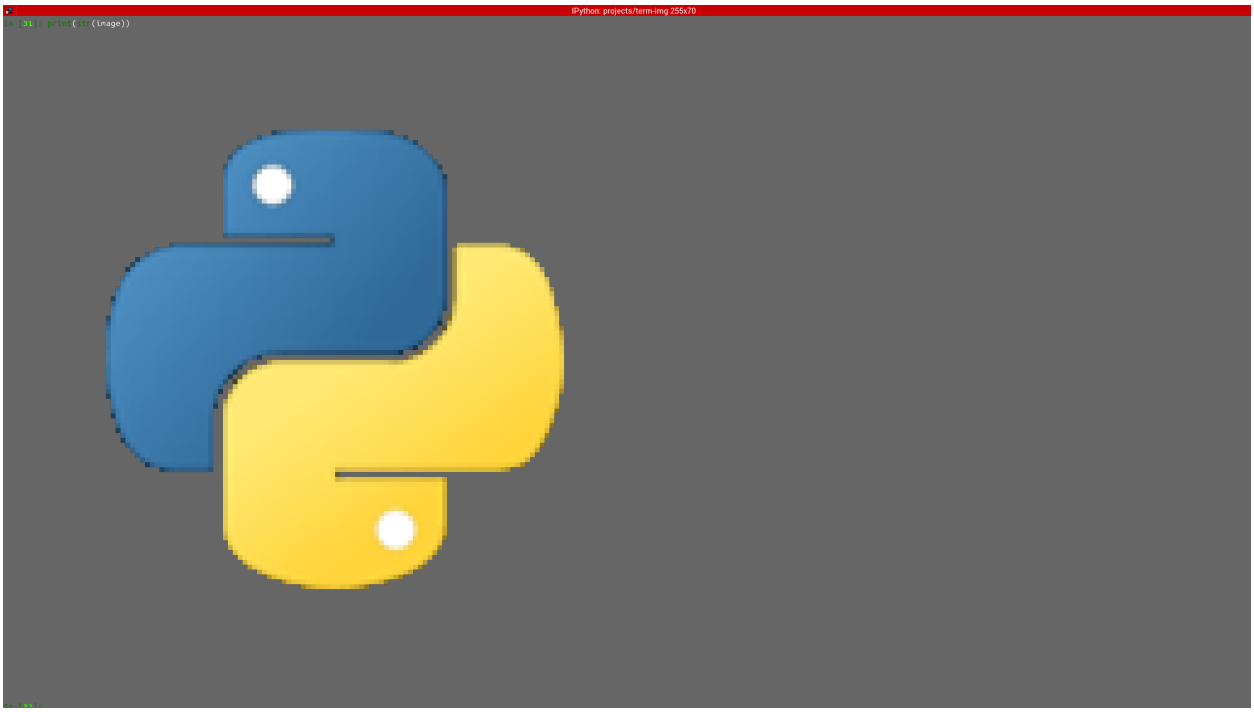
There are two ways to render an image:

1. Unformatted

```
str(image)
```

Renders the image without *padding/alignment* and with transparency enabled

The result should look like:



2. Formatted

Note: To see the effect of *alignment* in the steps below, please scale the image down using:

```
image.scale = 0.75
```

This simply sets the x-axis and y-axis *scales* of the image to `0.75`. We'll see more about this *later*.

Below are examples of formatted rendering:

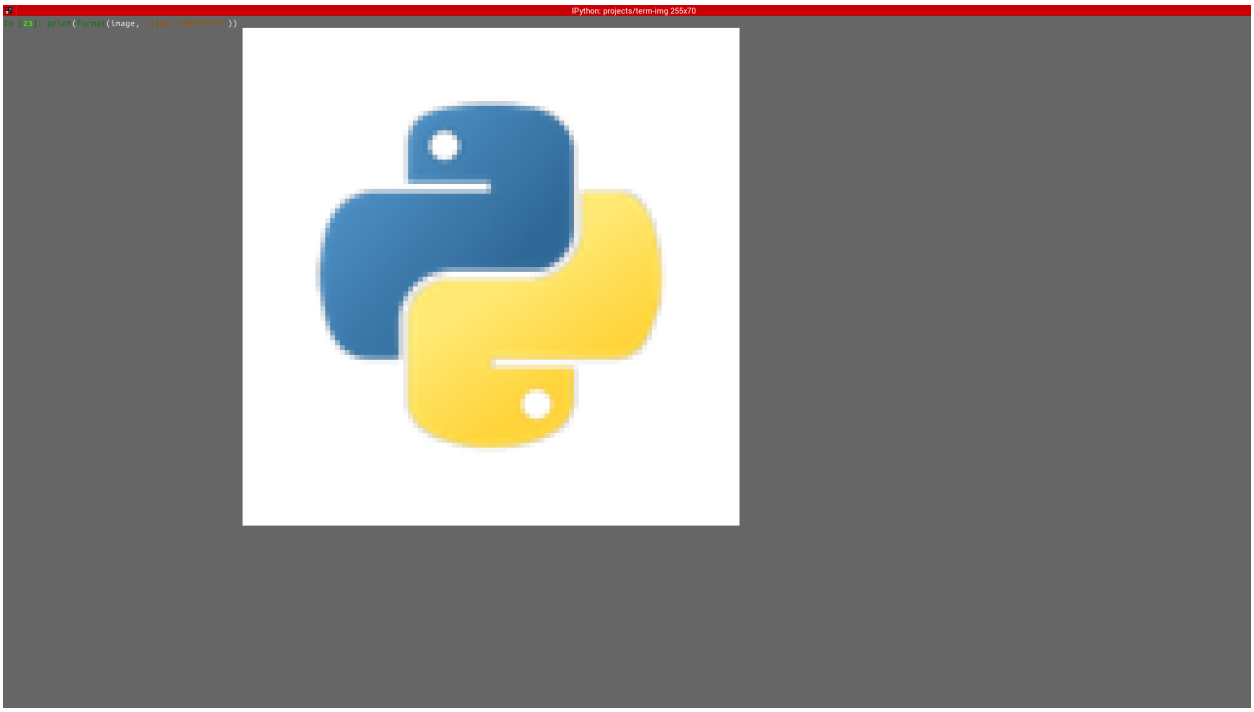
```
format(image, "|200.^70#ffffff")
```

Renders the image with:

- **center** *horizontal alignment*
- a *padding width* of **200** columns
- **top** *vertical alignment*
- a *padding height* of **70** lines
- transparent background replaced with a **white** (`#ffffff`) background

Note: You might have to reduce the padding width (200) and/or height (70) to something that'll fit into your terminal window, or increase the size of the terminlal window

The result should look like:

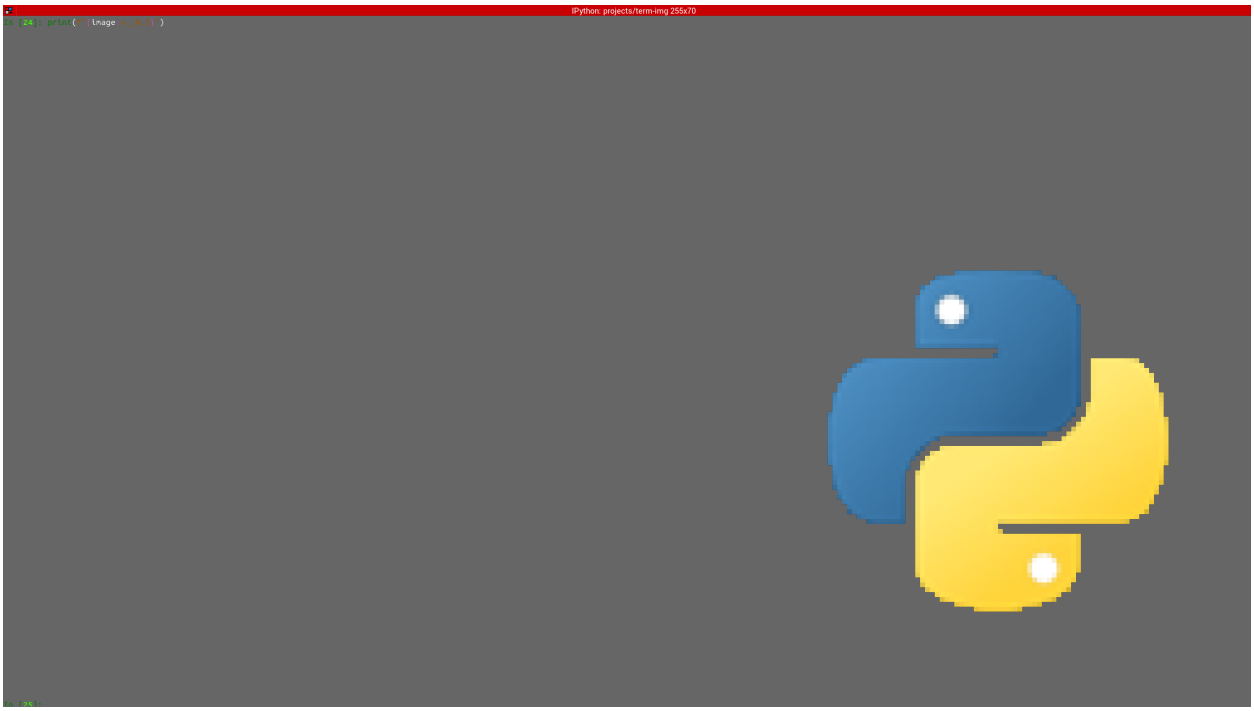



```
f"{image:>._#.5}"
```

Renders the image with:

- **right** *horizontal alignment*
- **automatic** *padding width* (the current *terminal width* minus *horizontal allowance*)
- **bottom** *vertical alignment*
- **automatic** *padding height* (the current *terminal height* minus *vertical allowance*)
- transparent background with **0.5** *alpha threshold*

The result should look like:



```
"{:1.1#}".format(image)
```

Renders the image with:

- **center** *horizontal alignment* (default)
- **no** horizontal *padding*, since 1 must be less than or equal to the image width
- **middle** *vertical alignment* (default)
- **no** vertical *padding*, since 1 is less than or equal to the image height
- transparency **disabled** (alpha channel is removed)

The result should look like:

You should also have a look at the complete *Image Format Specification*.

3.1.3 Drawing/Displaying an image to/in the terminal

There are two ways to draw an image to the terminal screen:

1. The *draw()* method

   ```
   image.draw()
   ```

   NOTE: *draw()* has various parameters for *alignment*/*padding*, transparency, animation control, etc.

2. Using `print()` with an image render output (i.e printing the rendered string)

   ```
   print(image)  # Uses str()
   ```

   OR

   ```
   print(f"{image:>200.^70#ffffff}")  # Uses format()
   ```

Note:

- For *animated* images, only the former animates the output, the latter only draws the **current** frame (see *seek()* and *tell()*).
- Also, the former performs size validation to see if the image will fit into the terminal, while the latter doesn't.

Important: All the examples above use automatic sizing and default *scale*.

3.1.4 Image size

The size of an image is the **unscaled** dimension with which an image is rendered.
The image size can be retrieved via the *size*, *width* and *height* properties.

The size of an image can be in either of two states:

1. Fixed

   In this state, the `size` property is a 2-tuple of integers, the `width` and `height` properties are integers.

2. Dynamic

   In this state,

   - the size with which the image is rendered is automatically calculated (based on the current *terminal size* or the image's original size) whenever the image is to be rendered.
   - the `size`, `width` and `height` properties evaluate to a *Size* enum member.

The size of an image can be set when creating an instance by passing an integer or a *Size* enum member to **either** the *width* **or** the *height* **keyword-only** parameter.
For whichever axis is given, the other axis is calculated **proportionally**.

Note:

1. The arguments can only be given **by keyword**.
2. If neither is given, the *FIT dynamic size* applies.
3. All methods of instantiation accept these arguments.

For example:

```
>>> from term_image.image import Size, from_file
>>> image = from_file("python.png")  # Dynamic FIT
>>> image.size is Size.FIT
True
>>> image = from_file("python.png", width=60)  # Fixed
>>> image.size
(60, 30)
>>> image.height
30
>>> image = from_file("python.png", height=56)  # Fixed
>>> image.size
(112, 56)
>>> image.width
112
>>> image = from_file("python.png", height=Size.FIT)  # Fixed FIT
>>> image.size
(136, 68)
>>> image = from_file("python.png", width=Size.FIT_TO_WIDTH)  # Fixed FIT_TO_WIDTH
>>> image.size
```

(continues on next page)

(continued from previous page)

```
(255, 128)
>>> image = from_file("python.png", height=Size.ORIGINAL)  # Fixed ORIGINAL
>>> image.size
(288, 144)
```

No size validation is performed i.e the resulting size might not fit into the terminal window

```
>>> image = from_file("python.png", height=68)  # Will fit, OK
>>> image.size
(136, 68)
>>> image = from_file("python.png", height=500)  # Will not fit, also OK
>>> image.size
(1000, 500)
```

An exception is raised when both *width* and *height* are given.

```
>>> image = from_file("python.png", width=100, height=100)
Traceback (most recent call last):
  .
  .
  .
ValueError: Cannot specify both width and height
```

The *width* and *height* properties are used to set the size of an image after instantiation.

```
>>> image = from_file("python.png")
>>> image.width = 56
>>> image.size
(56, 28)
>>> image.height
28
>>> image.height = 68
>>> image.size
(136, 68)
>>> image.width
136
>>> image.width = 200  # Even though the terminal can't contain the resulting height, the␣
↪size is still set
>>> image.size
(200, 100)
>>> image.width = Size.FIT
>>> image.size
(136, 69)
>>> image.height = Size.FIT_TO_WIDTH
>>> image.size
(255, 128)
>>> image.height = Size.ORIGINAL
>>> image.size
(288, 144)
```

Note: An exception is raised if the terminal size is too small to calculate a size.

The *size* property can only be set to a *Size* enum member, which results in a **dynamic** size.

```
>>> image = from_file("python.png")
>>> image.size = Size.FIT
>>> image.size is image.width is image.height is Size.FIT
True
>>> image.size = Size.FIT_TO_WIDTH
>>> image.size is image.width is image.height is Size.FIT_TO_WIDTH
True
>>> image.size = Size.ORIGINAL
>>> image.size is image.width is image.height is Size.ORIGINAL
True
```

Important:

1. The currently set *cell ratio* is also taken into consideration when calculating sizes.
2. There is a **default** 2-line *vertical allowance*, to allow for shell prompts or the likes.

Hint: See *set_size()* for extended sizing control.

3.1.5 Image scale

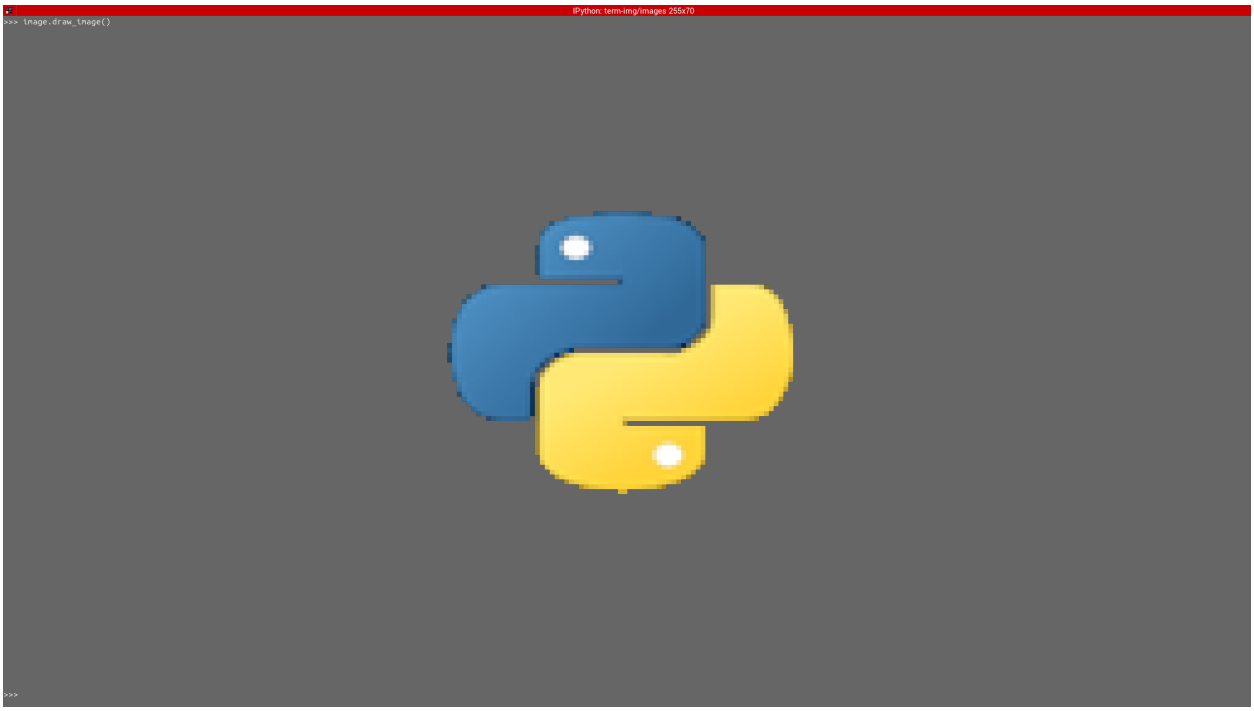
The scale of an image is the **fraction** of the size that'll actually be used to render the image.
A valid scale value is a `float` in the range `0 < x <= 1` i.e greater than zero and less than or equal to one.

The image scale can be retrieved via the properties *scale*, *scale_x* and *scale_y*.

The scale can be set at instantiation by passing a value to the *scale* **keyword-only** paramter.

```
>>> image = from_file("python.png", scale=(0.75, 0.6))
>>> image.scale
>>> (0.75, 0.6)
```

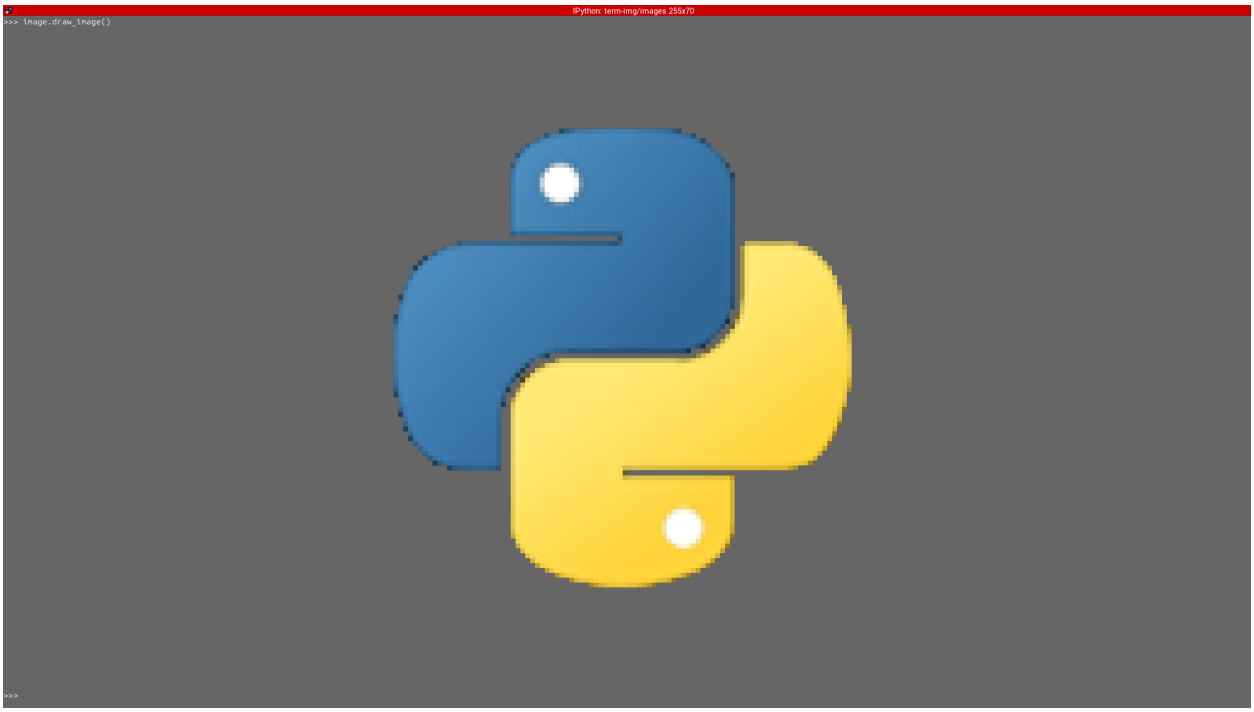
The drawn image (using `image.draw()`) should look like:



If the *scale* argument is ommited, the default scale `(1.0, 1.0)` is used.

```
>>> image = from_file("python.png")
>>> image.scale
>>> (1.0, 1.0)
```

The drawn image (using `image.draw()`) should look like:



The properties *scale*, *scale_x* and *scale_y* are used to set the scale of an image after instantiation.

`scale` accepts a tuple of two scale values or a single scale value.
`scale_x` and `scale_y` each accept a single scale value.

```
>>> image = from_file("python.png")
>>> image.scale = (.3, .56756)
>>> image.scale
(0.3, 0.56756)
>>> image.scale = .5
>>> image.scale
(0.5, 0.5)
>>> image.scale_x = .75
>>> image.scale
(0.75, 0.5)
>>> image.scale_y = 1.
>>> image.scale
(0.75, 1.0)
```

Finally, to explore more of the library's features and functionality, check out the *Reference* section.

3.2 Reference

Attention: Under Construction - There might be incompatible interface changes between minor versions of version zero!

If you want to use the library in a project while it's still on version zero, ensure you pin the dependency to a specific minor version e.g `>=0.4,<0.5`.

On this note, you probably also want to switch to the specific documentation for the version you're using (somewhere at the lower left corner of this page).

3.2.1 Core Library Definitions

The `term_image.image` subpackage defines the following:

Convenience Functions

These functions automatically detect the best supported render style for the current terminal.

Since all classes define a common interface, any operation supported by one image class can be performed on any other image class, except stated otherwise.

`term_image.image.`**`AutoImage`**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Convenience function for creating an image instance from a PIL image instance.

Returns
An instance of a subclass of *BaseImage*.

Return type
term_image.image.common.BaseImage

Same arguments and raised exceptions as the *BaseImage* class constructor.

term_image.image.**from_file**(*filepath*, ***kwargs*)

Convenience function for creating an image instance from an image file.

Returns
An instance of a subclass of *BaseImage*.

Return type
term_image.image.common.BaseImage

Same arguments and raised exceptions as *BaseImage.from_file()*.

term_image.image.**from_url**(*url*, ***kwargs*)

Convenience function for creating an image instance from an image URL.

Returns
An instance of a subclass of *BaseImage*.

Return type
term_image.image.common.BaseImage

Same arguments and raised exceptions as *BaseImage.from_url()*.

Image Classes

Class Hierachy:

- *ImageSource*
- *Size*
- *ImageMeta*
- *BaseImage*
  - *GraphicsImage*
    - *ITerm2Image*
    - *KittyImage*
  - *TextImage*
    - *BlockImage*

class term_image.image.**ImageSource**(*value*)

Bases: enum.Enum

Image source type.

Note: The values of the enumeration members are implementation details and might change at anytime. Any comparison should be by identity of the members themselves.

FILE_PATH = <hidden>

The instance was derived from a path to a local image file.

PIL_IMAGE = <hidden>

The instance was derived from a PIL image instance.

URL = <hidden>

The instance was derived from an image URL.

class term_image.image.**Size**(*value*)

Bases: enum.Enum

Enumeration for *automatic sizing*

AUTO = <hidden>

Equivalent to *ORIGINAL* if it will fit into the *available size*, else *FIT*.

FIT = <hidden>

The image size is set to fit optimally **within** the *available size*.

FIT_TO_WIDTH = <hidden>

The size is set such that the width is exactly the *available width*, regardless of the *cell ratio*.

ORIGINAL = <hidden>

The image size is set such that the image is rendered with as many pixels as the the original image consists of.

class term_image.image.**ImageMeta**(*name*, *bases*, *namespace*, ***kwargs*)

Bases: abc.ABCMeta

Type of all render style classes.

Note:

For all render style classes (instances of this class) defined **within** this package, `str(cls)` yeilds the same value as *cls.style*.

For render style classes defined **outside** this package (subclasses of those defined within this package), `str(cls)` is equivalent to `repr(cls)`.

---

property style

Name of the render style [category].

Returns

- The name of the render style [category] implemented by the invoking class, if defined **within** this package (`term_image`)
- `None`, if the invoking class is defined **outside** this package (`term_image`)

Return type
Optional[str]

Examples

For a class defined within this package:

```
>>> from term_image.image import KittyImage
>>> KittyImage.style
'kitty'
```

For a class defined outside this package:

```
>>> from term_image.image import KittyImage
>>> class MyImage(KittyImage): pass
>>> MyImage.style is None
True
```

Hint: Equivalent to `str(cls)` for all render style classes (instances of *ImageMeta*) defined **within** this package.

Note: It's allowed to set properties for *animated* images on non-animated ones, the values are simply ignored.

class term_image.image.**BaseImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: `object`

Base of all render styles.

Parameters

- **image** (*PIL.Image.Image*) – Source image.
- **width** (*Union[int, Size, None]*) – Can be
  - an `int`; horizontal dimension of the image, in columns.
  - a *Size* enum member.
- **height** (*Union[int, Size, None]*) – Can be
  - an `int`; vertical dimension of the image, in lines.
  - a *Size* enum member.
- **scale** (*Tuple[float, float]*) – The fraction of the size (on respective axes) to render the image with.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.

Propagates exceptions raised by *set_size()*, if *width* or *height* is given.

Note:

- If neither *width* nor *height* is given (or both are `None`), *FIT* applies.
- The image size is multiplied by the *scale* on respective axes when the image is *rendered*.
- For animated images, the seek position is initialized to the current seek position of the given image.

> **Attention:** This class cannot be directly instantiated. Image instances should be created from its subclasses.

property closed

Instance finalization status

Return type
bool

property frame_duration

Duration (in seconds) of a single frame for *animated* images

Setting this on non-animated images is simply ignored, no exception is raised.

Return type
float

property height

The **unscaled** height of the image.

Returns

- The image height (in lines), if the image size is *fixed*.
- A `Size` enum member; if the image size is *dynamic*.

Return type
Union[*Size*, int]

SETTABLE VALUES:

- a positive `int`; the image height is set to the given value and the width is set proportionally.
- a `Size` enum member; the image size is set as prescibed by the enum member.
- `None`; equivalent to `FIT`.

Setting this

- results in a *fixed size*.
- resets the recognized advanced sizing options to their defaults.

property is_animated

`True` if the image is *animated*. Otherwise, `False`.

property original_size

Size of the source image (in pixels)

property n_frames: int

The number of frames in the image

Return type
int

property rendered_height

The **scaled** height of the image.

Also the exact number of lines that the drawn image will occupy in a terminal.

Return type
int

property rendered_size

The **scaled** size of the image.

Also the exact number of columns and lines (respectively) that the drawn image will occupy in a terminal.

Return type
Tuple[int, int]

property rendered_width

The **scaled** width of the image.

Also the exact number of columns that the drawn image will occupy in a terminal.

Return type
int

property scale

Image *scale*

SETTABLE VALUES:

- A *scale value*; sets both axes.
- A `tuple` of two *scale values*; sets `(x, y)` respectively.

A scale value is a `float` in the range **0.0 < value <= 1.0**.

Return type
Tuple[float, float]

property scale_x

Horizontal *scale*

A scale value is a `float` in the range **0.0 < x <= 1.0**.

Return type
float

property scale_y

Vertical *scale*

A scale value is a `float` in the range **0.0 < y <= 1.0**.

Return type
float

property size

The **unscaled** size of the image.

Returns

- The image size, `(columns, lines)`, if the image size is *fixed*.
- A `Size` enum member, if the image size is *dynamic*.

Return type
Union[*Size*, Tuple[int, int]]

SETTABLE VALUES:

- A `Size` enum member; the image size is set as prescibed by the enum member.

Setting this

- implies *dynamic sizing* i.e the size is computed whenever the image is *rendered*.

- resets the recognized advanced sizing options to their defaults.

This is multiplied by the *scale* on respective axes when the image is *rendered*.

property source

The *source* from which the instance was initialized.

Return type
Union[PIL.Image.Image, str]

property source_type

The kind of *source* from which the instance was initialized.

Return type
ImageSource

property width

The **unscaled** width of the image.

Returns

- The image width (in columns), if the image size is *fixed*.
- A `Size` enum member; if the image size is *dynamic*.

Return type
Union[*Size*, int]

SETTABLE VALUES:

- a positive `int`; the image width is set to the given value and the height is set proportionally.
- a `Size` enum member; the image size is set as prescibed by the enum member.
- `None`; equivalent to `FIT`.

Setting this

- results in a *fixed size*.
- resets the recognized advanced sizing options to their defaults.

close()

Finalizes the instance and releases external resources.

- In most cases, it's not neccesary to explicity call this method, as it's automatically called when the instance is garbage-collected.
- This method can be safely called mutiple times.
- If the instance was initialized with a PIL image, the PIL image is never finalized.

draw(*h_align=None*, *pad_width=None*, *v_align=None*, *pad_height=None*, *alpha=0.1568627450980392*, ***, *scroll=False*, *animate=True*, *repeat=- 1*, *cached=100*, *check_size=True*, ***style*)

Draws an image to standard output.

Parameters

- **h_align** (*Optional[str]*) – Horizontal alignment ("left" / "<", "center" / "|" or "right" / ">"). Default: center.
- **pad_width** (*Optional[int]*) – Number of columns within which to align the image.

  - Excess columns are filled with spaces.
  - Must not be greater than the *available terminal width*.
  - Default: terminal width, minus horizontal allowance.
- **v_align** (*Optional[str]*) – Vertical alignment ("top"/"^", "middle"/"-" or "bottom"/"_"). Default: middle.
- **pad_height** (*Optional[int]*) – Number of lines within which to align the image.
  - Excess lines are filled with spaces.
  - Must not be greater than the *available terminal height*, **for animations**.
  - Default: terminal height, minus vertical allowance.
- **alpha** (*Optional[float, str]*) – Transparency setting.
  - If `None`, transparency is disabled (alpha channel is removed).
  - If a `float` (**0.0 <= x < 1.0**), specifies the alpha ratio **above** which pixels are taken as **opaque**. **(Applies to only text-based render styles)**.
  - If a string, specifies a color to replace transparent background with. Can be:
    * **"#"** -> The terminal's default background color (or black, if undetermined) is used.
    * A hex color e.g `ffffff`, `7faa52`.
- **scroll** (*bool*) – Only applies to non-animations. If `True`, allows the image's *rendered height* to be greater than the *available terminal height*.
- **animate** (*bool*) – If `False`, disable animation i.e draw only the current frame of an animated image.
- **repeat** (*int*) – The number of times to go over all frames of an animated image. A negative value implies infinite repetition.
- **cached** (*Union[bool, int]*) – Determines if *rendered* frames of an animated image will be cached (for speed up of subsequent renders of the same frame) or not.
  - If `bool`, it directly sets if the frames will be cached or not.
  - If `int`, caching is enabled only if the framecount of the image is less than or equal to the given number.
- **check_size** (*bool*) – If `False`, does not perform size validation for non-animations.
- **style** (*Any*) – Style-specific parameters. See each subclass for it's own usage.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- **ValueError** – Unable to convert image.
- **ValueError** – Image size or *scale* too small.
- ***term_image.exceptions.InvalidSizeError*** – The image's *rendered size* can not fit into the *available terminal size*.
- ***term_image.exceptions.StyleError*** – Unrecognized style-specific parameter(s).

- If *set_size()* was used to set the image size, the horizontal and vertical allowances (set when *set_size()* was called) are taken into consideration during size validation. If the size was set via another means or the size is *dynamic*, the default allowances apply.
- For **non-animations**, if the image size was set with :py:attr:term_image.image.Size.FIT_TO_WIDTH`, the image **height** is not validated and setting *scroll* is unnecessary.
- *animate*, *repeat* and *cached* apply to *animated* images only. They are simply ignored for non-animated images.
- For animations (i.e animated images with *animate* set to `True`):
  - *scroll* is ignored.
  - Image size and *padding height* are always validated, if set or given.
  - **with the exception of native animations provided by some render styles**.
- Animations, **by default**, are infinitely looped and can be terminated with **Ctrl+C** (`SIGINT`), raising `KeyboardInterrupt`.

classmethod from_file(*filepath*, ***kwargs*)

Creates an instance from an image file.

Parameters

- **filepath** (*str*) – Relative/Absolute path to an image file.
- **kwargs** (*Union[None, int, Tuple[float, float]]*) – Same keyword arguments as the class constructor.

Returns

A new instance.

Raises

- **TypeError** – *filepath* is not a string.
- **FileNotFoundError** – The given path does not exist.
- **IsADirectoryError** – Propagated from from `PIL.Image.open()`.
- **PIL.UnidentifiedImageError** – Propagated from from `PIL.Image.open()`.

Return type

term_image.image.common.BaseImage

Also Propagates exceptions raised or propagated by the class constructor.

classmethod from_url(*url*, ***kwargs*)

Creates an instance from an image URL.

Parameters

- **url** (*str*) – URL of an image file.
- **kwargs** (*Union[None, int, Tuple[float, float]]*) – Same keyword arguments as the class constructor.

Returns

A new instance.

Raises

- **TypeError** – *url* is not a string.

- **ValueError** – The URL is invalid.
- ***term_image.exceptions.URLNotFoundError*** – The URL does not exist.
- **PIL.UnidentifiedImageError** – Propagated from `PIL.Image.open()`.

Return type
term_image.image.common.BaseImage

Also propagates connection-related exceptions from `requests.get()` and exceptions raised or propagated by the class constructor.

Note: This method creates a temporary image file, but only after a successful initialization.

Proper clean-up is guaranteed except maybe in very rare cases.

To ensure 100% guarantee of clean-up, use the object as a *context manager*.

---

abstract classmethod is_supported()

Returns `True` if the render style or graphics protocol implemented by the invoking class is supported by the *active terminal*. Otherwise, `False`.

> **Attention:** Support checks for most (if not all) render styles require *querying* the *active terminal*, though **only the first time** they're executed.
>
> Hence, it's advisable to perform all neccesary support checks (call `is_supported()` on required subclasses) at an early stage of a program, before user input is required.

seek(*pos*)

Changes current image frame.

Parameters
pos (*int*) – New frame number.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.

Frame numbers start from 0 (zero).

classmethod set_render_method(*method=None*)

Sets the render method used by the instances of subclasses providing multiple render methods.

Parameters
method (*Optional[str]*) – The render method to be set or `None` for a reset (case-insensitive).

Raises

- **TypeError** – *method* is not a string or `None`.
- **ValueError** – the given method is not implmented by the invoking class (or class of the invoking instance).

See the **Render Methods** section in the description of the subclasses that implement such for their specific usage.

If called via:

- a class, sets the class-wide render method.
- an instance, sets the instance-specific render method.

If *method* is `None` and this method is called via:

- a class, the class-wide render method is reset to the default.
- an instance, the instance-specific render method is removed, so that it uses the class-wide render method thenceforth.

Any instance without a specific render method set uses the class-wide render method.

Note: *method* = `None` is always allowed, even if the render style doesn't implement multiple render methods.

---

set_size(*width=None*, *height=None*, *h_allow=0*, *v_allow=2*, *maxsize=None*)

Sets the image size with extended control.

Parameters

- **width** (*Optional[Union[int, term_image.image.common.Size]]*) – Can be
  - an `int`; horizontal dimension of the image, in columns.
  - a *Size* enum member.
- **height** (*Optional[Union[int, term_image.image.common.Size]]*) – Can be
  - an `int`; vertical dimension of the image, in lines.
  - a *Size* enum member.
- **h_allow** (*int*) – Horizontal allowance i.e minimum number of columns to leave unused.
- **v_allow** (*int*) – Vertical allowance i.e minimum number of lines to leave unused.
- **maxsize** (*Optional[Tuple[int, int]]*) – If given, as `(columns, lines)`, it's used instead of the terminal size.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- **ValueError** – Both *width* and *height* are specified.
- **ValueError** – The *available size* is too small for *automatic sizing*.
- ***term_image.exceptions.InvalidSizeError*** – *maxsize* is given and the resulting size will not fit into it.

If neither *width* nor *height* is given (or both are `None`), *FIT* applies.

If *width* or *height* is a *Size* enum member, *automatic sizing* applies as prescribed by the enum member.

When *FIT_TO_WIDTH* is given,

- size validation operations take it into consideration.
- *Vertical allowance* is nullified.

Allowances are ignored when *maxsize* is given.

Image formatting and size validation operations recognize and respect the horizontal and vertical allowances, until the image size is re-set.

Note: The size is checked to fit in only when *maxsize* is given along with a fixed *width* or *height* because `draw()` is generally not the means of drawing such an image and all rendering methods don't perform any sort of size validation.

If the validation is not desired, specify only one of *maxsize* and *width* or *height*, not both.

---

tell()

Returns the current image frame number.

class term_image.image.**GraphicsImage**(*image*, ***kwargs*)

Bases: `term_image.image.common.BaseImage`

Base of all render styles using terminal graphics protocols.

Raises

term_image.exceptions.StyleError – The *active terminal* doesn't support the render style.

See `BaseImage` for the description of the constructor.

Attention: This class cannot be directly instantiated. Image instances should be created from its subclasses.

class term_image.image.**TextImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: `term_image.image.common.BaseImage`

Base of all render styles using ASCII/Unicode symbols [with ANSI color codes].

See `BaseImage` for the description of the constructor.

Important: Instantiation of subclasses is always allowed, even if the current terminal does not [fully] support the render style.

To check if the render style is fully supported in the current terminal, use `is_supported()`.

> **Attention:** This class cannot be directly instantiated. Image instances should be created from its subclasses.

class term_image.image.**BlockImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: *term_image.image.common.TextImage*

A render style using unicode half blocks and ANSI 24-bit colour escape codes.

See *TextImage* for the description of the constructor.

classmethod is_supported()

Returns `True` if the render style or graphics protocol implemented by the invoking class is supported by the *active terminal*. Otherwise, `False`.

> **Attention:** Support checks for most (if not all) render styles require *querying* the *active terminal*, though **only the first time** they're executed.
>
> Hence, it's advisable to perform all neccesary support checks (call `is_supported()` on required subclasses) at an early stage of a program, before user input is required.

class term_image.image.**ITerm2Image**(*image*, ***kwargs*)

Bases: *term_image.image.common.GraphicsImage*

A render style using the iTerm2 inline image protocol.

See *GraphicsImage* for the complete description of the constructor.

Render Methods:

ITerm2Image provides two methods of *rendering* images, namely:

LINES (default)
Renders an image line-by-line i.e the image is evenly split across the number of lines it should occupy.

Pros:

- Good for use cases where it might be required to trim some lines of the image.

Cons:

- Image drawing is very slow on iTerm2 due to the terminal emulator's performance.

WHOLE
Renders an image all at once i.e the entire image data is encoded into one line of the *rendered* output, such that the entire image is drawn once by the terminal and still occupies the correct amount of lines and columns.

Pros:

- Render results are more compact (i.e less in character count) than with the `lines` method since the entire image is encoded at once.
- Image drawing is faster than with `lines` on most terminals.
- Smoother animations.

Cons:

- This method currently doesn't work well on iTerm2 and WezTerm when the image height is greater than the terminal height.

Note: The **LINES** method is the default only because it works properly in all cases, it's more advisable to use the **WHOLE** method except when the image height is greater than the terminal height or when trimming the image is required.

The render method can be set with *set_render_method()* using the names specified above.

Format Specification

See *Image Format Specification*.

```
[method] [ m {0 | 1} ] [ c {0-9} ]
```

- `method`: Render method override.

  Can be one of:

  - `L`: **LINES** render method (current frame only, for animated images).
  - `W`: **WHOLE** render method (current frame only, for animated images).
  - `N`: Native animation. Ignored when used with non-animated images, WEBP images or `ImageIterator`.

  Default: Current effective render method of the image.

- `m`: Cell content inter-mix policy (**Only supported in WezTerm**, ignored otherwise).
  - If the character after `m` is:
    - `0`, contents of cells in the region covered by the image will be erased.
    - `1`, the opposite, thereby allowing existing cell contents to show under transparent areas of the image.
  - If *absent*, defaults to `m0`.
  - e.g `m0`, `m1`.
- `c`: ZLIB compression level, for images re-encoded in PNG format.
  - 1 -> best speed, 9 -> best compression, 0 -> no compression.
  - This results in a trade-off between render time and data size/draw speed.
  - If *absent*, defaults to `c4`.
  - e.g `c0`, `c9`.

Attention: Currently supported terminal emulators include:

- iTerm2
- Konsole >= 22.04.0
- WezTerm

`JPEG_QUALITY: int = -1`

- `x < 0`, JPEG encoding is disabled.
- `0 <= x <= 95`, JPEG encoding is used, with the specified quality, for **most** non-transparent renders (at the cost of image quality).

Only applies when not reading directly from file.

By default, images are encoded in the PNG format (when not reading directly from file) but in some cases, higher compression might be desired. Also, JPEG encoding is significantly faster and can be useful to improve non-native animation performance.

Hint: The transparency status of some images can not be correctly determined in an efficient way at render time. To ensure the JPEG format is always used for a re-encoded render, disable transparency or set a background color.

`NATIVE_ANIM_MAXSIZE: int = 2097152`

Maximum size (in bytes) of image data for native animation.

`TermImageWarning` is issued (and shown **only the first time**, except a filter is set to do otherwise) if the image data size for a native animation is above this value.

This value can be altered but should be done with caution to avoid excessive memory usage.

`READ_FROM_FILE: bool = True`

- `True`, image data is read directly from file when possible and no image manipulation is required.
- `False`, images are always loaded and re-encoded, in the PNG format by default.

This is an optimization to reduce render times and is only applicable to the **WHOLE** render method, since the the **LINES** method inherently requires image manipulation.

Note: This setting does not affect animations, native animations are always read from file when possible and frames of non-native animations have to be loaded and re-encoded.

`classmethod clear()`

Clears all images on-screen.

Required and works only on Konsole, as text doesn't overwrite images.

draw(**args*, *method=None*, *mix=False*, *compress=4*, *native=False*, *stall_native=True*, ***kwargs*)

Draws an image to standard output.

Extends the common interface with style-specific parameters.

Parameters

- **args** – Positional arguments passed up the inheritance chain.
- **method** (*Optional[str]*) – Render method override. If `None` or not given, the current effective render method of the instance is used.
- **mix** (*bool*) – Cell content inter-mix policy (**Only supported in WezTerm**, ignored otherwise). If:
  - `False`, contents of cells within the region covered by the image are erased.
  - `True`, the opposite, thereby allowing existing text or image pixels to show under transparent areas of the image.
- **compress** (*int*) – ZLIB compression level, for images re-encoded in PNG format.

  An integer between 0 and 9: 1 -> best speed, 9 -> best compression, 0 -> no compression. This results in a trade-off between render time and data size/draw speed.
- **native** (*bool*) – If `True`, use native animation (if supported).
  - Ignored for non-animations.
  - *animate* must be `True`.
  - *alpha*, *repeat*, *cached* and *style* do not apply.
  - Always loops infinitely.
  - No control over frame duration.
  - Not all animated image formats are supported e.g WEBP.
  - The limitations of the **WHOLE** render method also apply.
  - Normal restrictions for rendered/padding height of animations do not apply.
- **stall_native** (*bool*) – Native animation execution control. If:
  - `True`, block until `SIGINT` (Ctrl+C) is recieved.
  - `False`, return as soon as the image is transmitted.
- **kwargs** – Keyword arguments passed up the inheritance chain.

Raises

term_image.exceptions.ITerm2ImageError – Native animation is not supported.

See the `draw()` method of the parent classes for full details, including the description of other parameters.

classmethod is_supported()

Returns `True` if the render style or graphics protocol implemented by the invoking class is supported by the *active terminal*. Otherwise, `False`.

> **Attention:** Support checks for most (if not all) render styles require *querying* the *active terminal*, though **only the first time** they're executed.
>
> Hence, it's advisable to perform all neccesary support checks (call `is_supported()` on required subclasses) at an early stage of a program, before user input is required.

class term_image.image.**KittyImage**(*image*, ***kwargs*)

Bases: *term_image.image.common.GraphicsImage*

A render style using the Kitty terminal graphics protocol.

See *GraphicsImage* for the complete description of the constructor.

Render Methods

KittyImage provides two methods of *rendering* images, namely:

LINES (default)
Renders an image line-by-line i.e the image is evenly split across the number of lines it should occupy.

Pros:

- Good for use cases where it might be required to trim some lines of the image.

WHOLE
Renders an image all at once i.e the entire image data is encoded into one line of the *rendered* output, such that the entire image is drawn once by the terminal and still occupies the correct amount of lines and columns.

Pros:

- Render results are more compact (i.e less in character count) than with the **LINES** method since the entire image is encoded at once.

The render method can be set with *set_render_method()* using the names specified above.

Format Specification

See *Image Format Specification*.

```
[method] [ z [index] ] [ m {0 | 1} ] [ c {0-9} ]
```

- `method`: Render method override.

  Can be one of:

  - `L`: **LINES** render method (current frame only, for animated images).
  - `W`: **WHOLE** render method (current frame only, for animated images).

  Default: Current effective render method of the image.

- `z`: Image/Text stacking order.
  - `index`: Image z-index. An integer in the **signed 32-bit range**.

    Images drawn in the same location with different z-index values will be blended if they are semi-transparent. If `index` is:

    * `>= 0`, the image will be drawn above text.
    * `< 0`, the image will be drawn below text.
    * `< -(2**31)/2`, the image will be drawn below cells with non-default background color.
  - `z` without `index` is currently only used internally.
  - If *absent*, defaults to `z0` i.e z-index zero.

  - e.g `z0`, `z1`, `z-1`, `z2147483647`, `z-2147483648`.

- `m`: Image/Text inter-mixing policy.
  - If the character after `m` is:
    * `0`, text within the region covered by the image will be erased, though text can be inter-mixed with the image after it's been drawn.
    * `1`, text within the region covered by the image will NOT be erased.
  - If *absent*, defaults to `m0`.
  - e.g `m0`, `m1`.
- `c`: ZLIB compression level.
  - 1 -> best speed, 9 -> best compression, 0 -> no compression.
  - This results in a trade-off between render time and data size/draw speed.
  - If *absent*, defaults to `c4`.
  - e.g `c0`, `c9`.

Attention: Currently supported terminal emulators include:

- Kitty >= 0.20.0.
- Konsole >= 22.04.0.

static clear(*all=True*)

Clears images on-screen.

Parameters

all (*bool*) – If `False`, clears only the images intersecting with the cursor. Otherwise, clears all images currently on the screen.

draw(**args*, *method=None*, *z_index=0*, *mix=False*, *compress=4*, ***kwargs*)

Draws an image to standard output.

Extends the common interface with style-specific parameters.

Parameters

- **args** – Positional arguments passed up the inheritance chain.
- **method** (*Optional[str]*) – Render method override. If `None` or not given, the current effective render method of the instance is used.
- **z_index** (*Optional[int]*) – The stacking order of images and text **for non-animations**.

  Images drawn in the same location with different z-index values will be blended if they are semi-transparent. If *z_index* is:

  - >= `0`, the image will be drawn above text.
  - < `0`, the image will be drawn below text.
  - < `-(2**31)/2`, the image will be drawn below cells with non-default background color.

- `None`, internal use only, mentioned for the sake of completeness.

To inter-mixing text with an image, see the *mix* parameter.

- **mix** (*bool*) – Image/Text inter-mixing policy **for non-animations**. If:
  - `True`, text within the region covered by the image will NOT be erased.
  - `False`, text within the region covered by the image will be erased, though text can be inter-mixed with the image after it's been drawn.
- **compress** (*int*) – ZLIB compression level.

  An integer between 0 and 9: 1 -> best speed, 9 -> best compression, 0 -> no compression. This results in a trade-off between render time and data size/draw speed.
- **kwargs** – Keyword arguments passed up the inheritance chain.

See the `draw()` method of the parent classes for full details, including the description of other parameters.

classmethod is_supported()

Returns `True` if the render style or graphics protocol implemented by the invoking class is supported by the *active terminal*. Otherwise, `False`.

> **Attention:** Support checks for most (if not all) render styles require *querying* the *active terminal*, though **only the first time** they're executed.
>
> Hence, it's advisable to perform all neccesary support checks (call `is_supported()` on required subclasses) at an early stage of a program, before user input is required.

class term_image.image.**ImageIterator**(*image*, *repeat=- 1*, *format=''*, *cached=100*)

Bases: `object`

Effeciently iterate over *rendered* frames of an *animated* image

Parameters

- **image** (`BaseImage`) – Animated image.
- **repeat** (*int*) – The number of times to go over the entire image. A negative value implies infinite repetition.
- **format** (*str*) – The *format specifier* to be used to format the rendered frames (default: auto).
- **cached** (*Union[bool, int]*) – Determines if the *rendered* frames will be cached (for speed up of subsequent renders) or not.
  - If `bool`, it directly sets if the frames will be cached or not.
  - If `int`, caching is enabled only if the framecount of the image is less than or equal to the given number.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.

- *term_image.exceptions.StyleError* – Invalid style-specific format specifier.

- If *repeat* equals `1`, caching is disabled.
- The iterator has immediate response to changes in the image size and *scale*.
- If the image size is *dynamic*, it's computed per frame.
- The number of the last yielded frame is set as the image's seek position.
- Directly adjusting the seek position of the image doesn't affect iteration. Use *ImageIterator.seek()* instead.
- After the iterator is exhausted, the underlying image is set to frame `0`.

property loop_no

Iteration repeat countdown

Changes on the first iteration of each loop, except for infinite iteration where it's always `-1`.

close()

Closes the iterator and releases resources used.

Does not reset the frame number of the underlying image.

Note: This method is automatically called when the iterator is exhausted or garbage-collected.

---

seek(*pos*)

Sets the frame number to be yielded on the next iteration without affecting the repeat count.

Parameters
pos (*int*) – Next frame number.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- *term_image.exceptions.TermImageError* – Iteration has not yet started or the iterator is exhausted/closed.

Frame numbers start from `0` (zero).

Context Management Protocol Support

`BaseImage` instances are context managers i.e they can be used with the `with` statement as in:

```
with from_url(url) as image:
    ...
```

Using an instance as a context manager more surely guarantees **object finalization** (i.e clean-up/release of resources), especially for instances with URL sources (see *`BaseImage.from_url()`*).

Iteration Support

Animated `BaseImage` instances are iterable i.e they can be used with the `for` statement (and other means of iteration such as unpacking) as in:

```
for frame in from_file("animated.gif"):
    ...
```

Subsequent frames of the image are yielded on subsequent iterations.

Note:

- `iter(anim_image)` returns an *`ImageIterator`* instance with a repeat count of *1*, hence caching is disabled.
- The frames are unformatted and transparency is enabled i.e as returned by `str(image)`.

For more extensive or custom iteration, use *`ImageIterator`* directly.

3.2.2 Custom Exceptions

The `term_image.exceptions` module defines the following:

exception `term_image.exceptions.`**TermImageWarning**

Bases: `UserWarning`

Package-specific warning category.

exception `term_image.exceptions.`**TermImageError**

Bases: `Exception`

Exception baseclass. Raised for generic errors.

exception `term_image.exceptions.`**URLNotFoundError**

Bases: `FileNotFoundError`, *`term_image.exceptions.TermImageError`*

Raised for 404 errors.

exception `term_image.exceptions.`**InvalidSizeError**

Bases: `ValueError`, *`term_image.exceptions.TermImageError`*

Raised for invalid image sizes.

exception term_image.exceptions.**StyleError**

Bases: *term_image.exceptions.TermImageError*

Baseclass of style-specific exceptions.

Never raised for errors pertaining to image classes defined in this package. Instead, the exception subclass specific to each image class is raised.

Only raised for subclasses of *BaseImage* defined outside this package (which are not subclasses of any other image class defined in this package).

Being the baseclass of all style-specific exceptions, it can be used be used to handle any style-specific error, regardless of the render style it originated from.

exception term_image.exceptions.**GraphicsImageError**

Bases: *term_image.exceptions.StyleError*

Raised for errors specific to *GraphicsImage* and its subclasses defined outside this package.

exception term_image.exceptions.**TextImageError**

Bases: *term_image.exceptions.StyleError*

Raised for errors specific to *TextImage* and its subclasses defined outside this package.

exception term_image.exceptions.**BlockImageError**

Bases: *term_image.exceptions.TextImageError*

Raised for errors specific to *BlockImage* and its subclasses defined outside this package.

exception term_image.exceptions.**ITerm2ImageError**

Bases: *term_image.exceptions.GraphicsImageError*

Raised for errors specific to *ITerm2Image* and its subclasses defined outside this package.

exception term_image.exceptions.**KittyImageError**

Bases: *term_image.exceptions.GraphicsImageError*

Raised for errors specific to *KittyImage* and its subclasses defined outside this package.

3.2.3 Utilities

Every mention of *active terminal* in this module refers to the first terminal device discovered.

The following streams/files are checked in the following order of priority (along with the rationale behind the ordering):

- `STDOUT`: Since it's where images will most likely be drawn.
- `STDIN`: If output is redirected to a file or pipe and the input is a terminal, then using it as the *active terminal* should give the expected result i.e the same as when output is not redirected.
- `STDERR`: If both output and input are redirected, it's usually unlikely for errors to be.
- `/dev/tty`: Finally, if all else fail, fall back to the process' controlling terminal, if any.

The first one that is ascertained to be a terminal device is used for all terminal queries and terminal size computations.

Note: If none of the streams/files is a terminal device, then a warning is issued and affected functionality disabled.

Terminal Queries

Some functionalities of this library require the aquisition of certain information from the *active terminal*. A single iteration of this aquisition procedure is called a **query**.

A query involves three major steps:

1. Clear all unread input from the terminal
2. Write to the terminal
3. Read from the terminal

For this procedure to be successful, it must not be interrupted.

About #1
: If the program is expecting input, use *`read_tty()`* (simply calling it without any argument is enough) to read all currently unread input (**without blocking**) just before any operation involving a query.

About #2 and #3
: After sending a request to the terminal, its response is awaited. The default wait time is **0.1 seconds** but can be changed using *`set_query_timeout()`*.

  If the program includes any other function that could write to the terminal OR especially, read from the terminal or modify it's attributes, while a query is in progress, decorate it with *`lock_tty()`* to ensure it doesn't interfere.

  For example, the TUI included in this package (i.e `term_image`) uses urwid which reads from the terminal using `urwid.raw_display.Screen.get_available_raw_input()`. To prevent this method from interfering with terminal queries, it is wrapped thus:

```
urwid.raw_display.Screen.get_available_raw_input = lock_tty(
    urwid.raw_display.Screen.get_available_raw_input
)
```

  Also, if the *active terminal* is not the controlling terminal of the process using this library (e.g output is redirected to another terminal), ensure no process that can interfere with a query (e.g a shell) is currently running in the active terminal.
  For instance, such a process can be temporarily put to sleep.

List of features that use terminal queries

In parentheses are the outcomes when the terminal doesn't support queries or when queries are disabled.

- *Auto Cell Ratio* (determined to be unsupported)
- Support checks for *Graphics-based Render Styles* (determined to be unsupported)
- Auto background color (black is used)
- Alpha blend for pixels above the alpha threshold in transparent renders with *Text-based Render Styles* (black is used)
- Workaround for ANSI background colors in text-based renders on the Kitty terminal (the workaround is disabled)

Note: This list might not always be complete. In case you notice

- any difference with any unlisted feature when terminal queries are enabled versus when disabled, or
- a behaviour different from the one specified for the listed features, when terminal queries are disabled,

please open an issue here.

The `term_image.utils` module provides the following public definitions.

Attention: Any other definition in this module should be considered part of the private interface and can change without notice.

`term_image.utils.`**`lock_tty`**(*func*)

Synchronizes access to the *active terminal*.

Parameters
`func` (*`Callable`*) – The function to be wrapped.

When any decorated function is called, a re-entrant lock is acquired by the current process or thread and released after the call, such that any other decorated function called within another thread or subprocess has to wait till the lock is fully released (i.e has been released as many times as acquired) by the current process or thread.

Note:

It automatically works across parent-/sub-processes, started directly or indirectly via `multiprocessing.Process` (or a subclass of it) and their threads.

Important: It only works if `multiprocessing.synchronize` is supported on the host platform. If not supported, a warning is issued when starting a subprocess.

`term_image.utils.`**`read_tty`**(*more=<function <lambda>>*, *timeout=None*, *min=0*, ***, *echo=False*)

Reads input directly from the *active terminal* with/without blocking.

Parameters

- **`more`** (*`Callable[[bytearray], bool]`*) – A callable, which when passed the input recieved so far, as a *bytearray* object, returns a boolean. If it returns:
  - `True`, more input is waited for.
  - `False`, the input recieved so far is returned immediately.
- **`timeout`** (*`Optional[float]`*) – Time limit for awaiting input, in seconds.
- **`min`** (*`int`*) – Causes to block until at least the given number of bytes have been read.
- **`echo`** (*`bool`*) – If `True`, any input while waiting is printed unto the screen. Any input before or after calling this function is not affected.

Returns
The input read (empty, if *min* == `0` (default) and no input is recieved before *timeout* is up).

Return type
Optional[bytes]

If *timeout* is `None` (default), all available input is read without blocking.

If *timeout* is not `None` and:

- *timeout* < `0`, it's infinite.
- *min* > `0`, input is waited for until at least *min* bytes have been read.

  After *min* bytes have been read, the following points apply with *timeout* being the leftover of the original *timeout*, if not yet used up.
- *more* is not given, input is read or waited for until *timeout* is up.
- *more* is given, input is read or waited for until `more(input)` returns `False` or *timeout* is up.

Upon return or interruption, the *active terminal* is **immediately** restored to the state in which it was met.

Note: Currently works on UNIX only, returns `None` on any other platform or when there is no *active terminal*.

`term_image.utils.`**`set_query_timeout`**`(`*`timeout`*`)`

Sets the global timeout for *Terminal Queries*.

Parameters

`timeout` (*`float`*) – Time limit for awaiting a response from the terminal, in seconds.

Raises

- **`TypeError`** – *timeout* is not a float.
- **`ValueError`** – *timeout* is less than or equal to zero.

`term_image.utils.`**`DISABLE_QUERIES: bool = False`**

If `True`, *Terminal Queries* are disabled, thereby affecting all *dependent features*.

`term_image.utils.`**`SWAP_WIN_SIZE: bool = False`**

A workaround for some terminal emulators (e.g older VTE-based ones) that wrongly report window dimensions swapped.

If `True`, the dimensions reported by the terminal emulator are swapped.

This setting affects *Auto Cell Ratio* computation.

3.2.4 Top-Level Definitions

`term_image.`**`set_cell_ratio`**`(`*`ratio`*`)`

Sets the global *cell ratio*.

Parameters

`ratio` (*`Union[float, term_image.AutoCellRatio]`*) – Can be one of the following values.

- A positive `float` value.
- *`AutoCellRatio.FIXED`*, the ratio is immediately determined from the *active terminal*.
- *`AutoCellRatio.DYNAMIC`*, the ratio is determined from the *active terminal* whenever *`get_cell_ratio()`* is called, though with some caching involved, such that the ratio is re-determined only if the terminal size changes.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- ***term_image.exceptions.TermImageError*** – Auto cell ratio is not supported in the *active terminal* or on the current platform.

This value is taken into consideration when setting image sizes for **text-based** render styles, in order to preserve the aspect ratio of images drawn to the terminal.

Note: Changing the cell ratio does not automatically affect any image that has a *fixed size*. For a change in cell ratio to take effect, the image's size has to be re-set.

Attention: See *Auto Cell Ratio* for details about the auto modes.

term_image.**get_cell_ratio**()

Returns the global *cell ratio*.

See *set_cell_ratio()*.

class term_image.**AutoCellRatio**(*value*)

Bases: enum.Enum

Values for setting *Auto Cell Ratio*.

is_supported: Optional[bool] = None

Auto cell ratio support status. Can be

- None -> support status not yet determined
- True -> supported
- False -> not supported

Can be explicitly set when using auto cell ratio but want to avoid the support check in a situation where the support status is foreknown. Can help to avoid being wrongly detected as unsupported on a *queried* terminal that doesn't respond on time.

For instance, when using multiprocessing, if the support status has been determined in the main process, this value can simply be passed on to and set within the child processes.

FIXED

DYNAMIC

See *set_cell_ratio()*.

3.2.5 Render Styles

A render style is a specific implementation of representing or drawing images in a terminal emulator and each is implemented as a class.

All render styles are designed to share a common interface (with some styles having extensions), making the usage of one class directly compatibile with another, except when using style-specific features.

Hence, the covenience functions *AutoImage*, *from_file()* and *from_url()* provide a means of render-style-independent usage of the library.
These functions automatically detect the best render style supported by the *active terminal*.

There a two categories of render styles:

Text-based Render Styles

Represent images using ASCII or Unicode symbols, and in some cases, with ANSI colour escape codes.

Classes for render styles in this category are subclasses of *TextImage*. These include:

- *BlockImage*

Graphics-based Render Styles

Represent images with actual pixels, using terminal graphics protocols.

Classes for render styles in this category are subclasses of *GraphicsImage*. These include:

- *KittyImage*
- *ITerm2Image*

3.2.6 Auto Cell Ratio

When using **auto cell ratio** (in either mode), it's important to note that some (not all) terminal emulators (e.g VTE-based ones) might have to be queried. **See** *Terminal Queries*.

If the program will never expect any useful input, **particularly while an image's size is being set/calculated** (for an image with *dynamic size*, while it's being rendered or its *rendered_size*, *rendered_width* or *rendered_height* property is invoked), then using `DYNAMIC` mode is OK.

Otherwise i.e if the program will be expecting input, use `FIXED` mode and use *utils.read_tty()* to read all currently unread input just before calling *set_cell_ratio()*.

Note: This concerns **text-based** render styles only (see the sub-section above).

3.2.7 Image Format Specification

```
[h_align] [width] [ . [v_align] [height] ] [ # [threshold | bgcolor] ] [ + {style} ]
```

Note:

- The spaces are only for clarity and not included in the syntax.
- Fields within `[ ]` are optional.
- Fields within `{ }` are required, though subject to any enclosing `[ ]`.
- `|` implies mutual exclusivity.
- If the `.` is present, then at least one of `v_align` and `height` must be present.
- `width` and `height` are in units of columns and lines repectively.
- If the *padding width* or *padding height* is less than or equal to the image's *rendered width* or *rendered height* respectively, the padding has **no effect**.

- `h_align`: This can be one of:
  - `<` → left
  - `|` → center
  - `>` → right
  - *Default* → center
- `width`: padding width
  - Positive integer
  - *Default*: *terminal width* minus *horizontal allowance*
- `v_align`: This can be one of:
  - `^` → top
  - `-` → middle
  - `_` → bottom
  - *Default* → middle
- `height`: padding height
  - Positive integer
  - *Default*: *terminal height* minus *vertical allowance*
- `#`: Transparency setting:
  - *Default*: transparency is enabled with the default *alpha threshold*.
  - `threshold`: *alpha threshold* e.g `.0`, `.325043`, `.99999`.
    - The value must be in the range **0.0 <= threshold < 1.0**.
    - **Applies to only text-based render styles** e.g. *`BlockImage`*.
  - `bgcolor`: Color to replace transparent background with. Can be:
    - `#` -> The terminal's default background color (or black, if undetermined) is used.

        * A hex color e.g `ffffff`, `7faa52`.
    - If neither `threshold` nor `bgcolor` is present, but `#` is present, transparency is disabled (alpha channel is removed).
- `style`: Style-specific format specifier.

  See each render style in *Image Classes* for its own specification, if it defines.

  `style` can be broken down into `[parent] [current]`, where `current` is the spec defined by a class and `parent` is the spec defined by a parent of that class. `parent` can in turn be **recursively** broken down as such.

See *Formatted rendering* for examples.

3.3 Known Issues

1. Drawing of images and animations doesn't work completely well with Python for windows (tested in Windows Terminal and MinTTY).
   - **Description:** Some lines of the image seem to extend beyond the number of columns that they should normally occupy by one or two columns.

     This behaviour causes animations to go bizzare when lines extend beyond the width of the terminal emulator.
   - **Comment:** First of all, the issue seems to caused by the layer between Python and the terminal emulators (i.e the PTY implementation in use) which "consumes" the escape sequences used to display images.

     It is neither a fault of this library nor of the terminal emulators, as drawing of images and animations works properly with WSL within Windows Terminal.
   - **Solution:** A workaround is to leave some **horizontal allowance** of **at least two columns** to ensure the image never reaches the right edge of the terminal.

     This can be achieved in the library using the *h_allow* parameter of `set_size()`.
2. Some animations with the **kitty** render style within the **Kitty terminal emulator** might be glitchy at the moment.
   - **Description:** When the **LINES** render method is used, lines of the image might intermittently disappear. When the **WHOLE** render method is used, the entire image might intermitently dissapear.
   - **Comment:** This is due to the fact that drawn each frame requires clearing the previous frame off the screen, since the terminal would otherwise blend subsequent frames. Not clearing previous frames would break transparent animations and result in a performance lag that gets worse over time.
   - **Solution:** Plans are in motion to implement support for native animations i.e utilizing the animation features provided by the protocol (See #40).

3.4 Planned Features

In no particular order:

- Performance improvements
- Support for more terminal graphics protocols (See #23)
- More text-based render styles (See #57)
- Support for terminal emulators with 256 colors, 8 colors and no color (See #61)
- Support for terminal emulators without Unicode support (See #58, #60)

- Support for fbTerm
- Support for open file objects and the `Pathlike` interface
- Determination of frame duration per frame during animations and image iteration
- Multithreaded animation
- Kitty image ID (See #40)
- Kitty native animation (See #40)
- Framing formatting option
- Image zoom and pan functionalities
- Setting images to their original size
- Key-value pair format specification
- Specifiy key to end animation
- Drawing images to an alternate output
- Use `termpile` for URL-sourced images
- Source images from raw pixel data
- IPython Extension
- Addition of urwid widgets for displaying images
- etc…

CHAPTER

FOUR

IMAGE VIEWER

Attention: Under Construction - There might be incompatible changes (particularly in the CLI options and configuration) between minor versions of version zero!

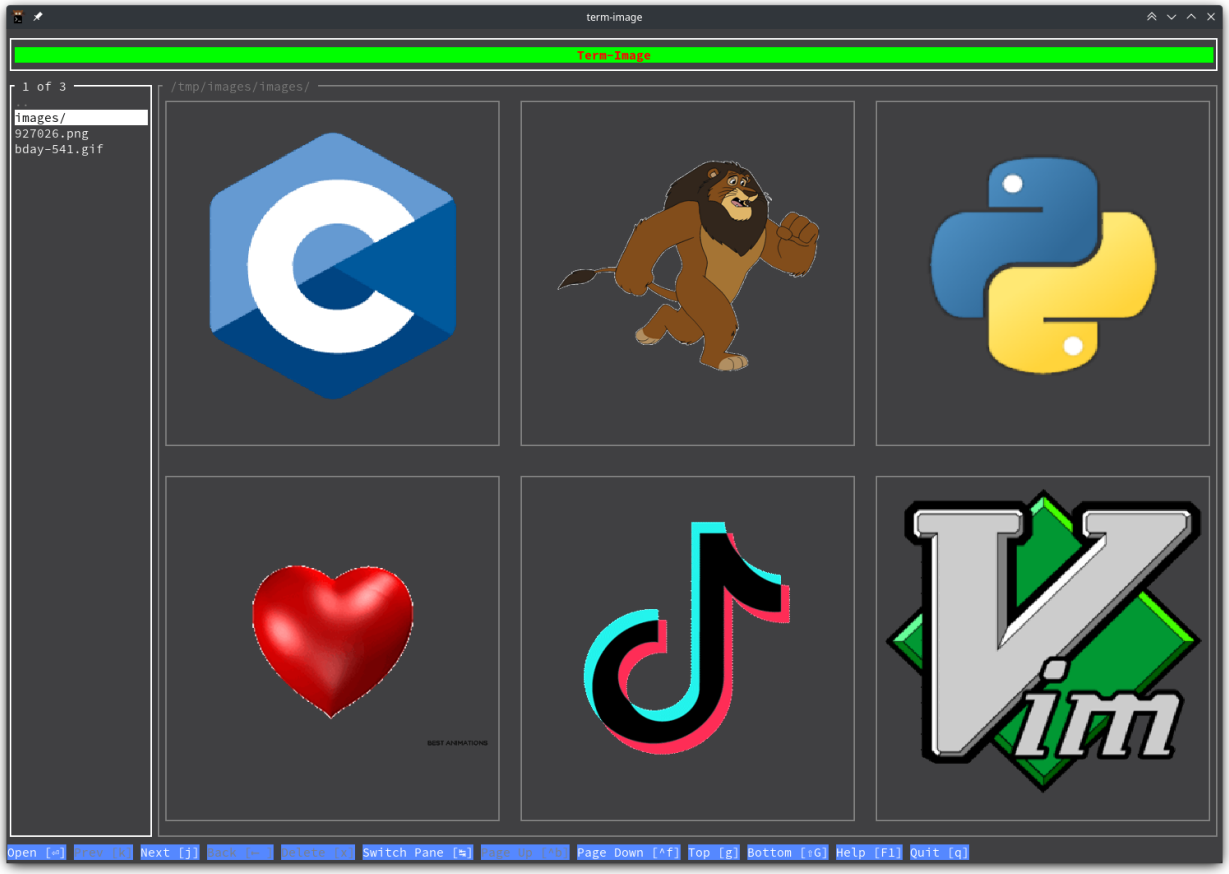
If you want to use the viewer in a project while it's still on version zero, ensure you pin the dependency to a specific minor version e.g `>=0.4,<0.5`.

On this note, you probably also want to switch to the specific documentation for the version you're using (somewhere at the lower left corner of this page).

4.1 Text-based User Interface

The TUI is developed using urwid.

4.1.1 Demo



See a demo video (*recorded at normal speed and not sped up*).

4.1.2 UI Components

The UI consists of various areas which are each composed using one or more widgets.
The components of the UI might change depending on the current *context* and some *actions*.

The following are the key components that make up the UI.

- **Banner**:
  - At the top of the UI.
  - Fixed height of 4 lines.
  - Contains the project title with a surrounding color fill and a line-box decoration.
  - Hidden in full image views.
- **Viewer**:
  - Immediately below the title banner.
  - Consists of two sub-components (described below) arranged horizontally: * Menu * View
- **Menu**:

  - Sub-component of the *viewer* to the left.
  - Fixed width of 20 columns.
  - Contains a list of image and directory entries which can be scrolled through.
  - Used to scroll through images in a directory and navigate back and forth through directories, among other actions.
- **View**:
  - Sub-component of the *viewer* to the right.
  - Images are displayed in here.
  - The content can be one of these two, depending on the type of item currently selected in the *menu*: * An image: When the item selected in the menu is an image. * An image grid: When the item selected in the menu is a directory.
  - The *view* component can also be used to scroll through images.
- **Notification Bar**:
  - Immediately above the *Action/Key Bar*.
  - Notifications about various events are displayed here.
  - Hidden in full image views.
  - Hidden in all views, in QUIET mode (`--quiet`).
- **Action/Key Bar**:
  - Contains a list of *actions* in the current *context*.
  - Each action has the symbol of the assigned key beside its name.
  - If the actions are too much to be listed on one line, the bar can be expanded/collapsed using the key indicated at the far right.
- **Overlays**:
  - These are used for various purposes such as help menu, confirmations, etc.
  - They are shown only when certain actions are triggered.
- **Info Bar**:
  - Used for debugging.
  - This is a 1-line bar immediately above the action/key bar.
  - Only shows (in all views) when the `--debug` option is specified.

Full/Maximized image views consist of only the *view* and *action/key bar* components.

4.1.3 Contexts

A context is simply a set of *actions*.

The active context might change due to one of these:

- Triggering certain *actions*.
- Change of *viewer* sub-component (i.e *menu* or *view*) in focus.
- Type of menu entry selected.
- An overlay is shown.

The active context determines which actions are available and displayed in the *action/key bar* at the bottom of the UI.

The following are the contexts available:

- **global**: The actions in this context are available when any other context is active, with a few exceptions.
- **menu**: This context is active when the *menu* UI component is in focus and non-empty.
- **image**: This context is active if the *view* UI component is in focus and was switched to (from the *menu*) while an image entry was selected.
- **image-grid**: This context is active if the *view* UI component is in focus and was switched to (from the *menu*) while a directory entry was selected.
- **full-image**: This context is active when an image entry is maximized from the `image` context (using the `Maximize` action) or from the `menu` context using the `Open` action.
- **full-grid-image**: This context is active when an image grid cell is maximized from the `image-grid` context (using the `Open` action).
- **confirmation**: This context is active only when specific actions that require confirmation are triggered e.g the `Delete` action in some contexts.
- **overlay**: This context is active only when an overlay UI component (e.g the help menu) is shown.

4.1.4 Actions

An action is a single entry in a *context*, it represents a functionality available in that context.
An action has the following defining properties:

- **name**: The name of the action.
- **key**: The key/combination used to trigger the action.
- **symbol**: A string used to represent the *key*.
- **description**: A brief description of what the action does.
- **visibility**: Determines if the action is displayed in the *action/key bar* or not.
- **state**: Determines if the action is enabled or not. * If an action is disabled, pressing its *key* will trigger the terminal bell.

Note: All contexts and their actions (with default properties) can be found here.

4.2 Configuration

The configuration is divided into the following categories:

- Options
- Keybindings

A configuration file is written in JSON format, using a *partial config* style i.e only the fields to be modified need to be present in the config file.

By default, `term-image` searches the following locations, in the specified order, for `$DIR/term_image/config.json` (a file named `config.json` within a `term_image` directory).

- All valid directories specified in the `XDG_CONFIG_DIRS` enviroment variable, **in reverse order** or `/etc/xdg` if not set.
- The directory specified in the `XDG_CONFIG_HOME` enviroment variable or `~/.config` if not set (where ~ is the current user's home directory).

If multiple config files are found (i.e in different locations), they're applied on top of one another **in the order in which they were found**. Hence, a field present in the latter, if valid, will override the same field also present in the former.

An alternative config file can be specified per-session using the `--config` command-line option.

To use the default configuration and not load any config file, use the `--no-config` command-line option.

Hint: `term-image` performs [quite] thorough validation on the values specified in a config file and reports any errors. To see information about how the errors are resolved (if resolvable), use the `-v/--verbose` command-line option.

This is a sample config file with all options and keybindings at their defaults. Note that **this is only for reference**, using any field within it as-is has no effect.

4.2.1 Config Options

These are fields whose values control various behaviours of the viewer.
Any option with a "[*]" after its description will be used only when a corresponding command-line option is either not specified or has an invalid value.

They are as follows:

anim cache
: The maximum frame count of an image for which frames will be cached during animation. [*]

  - Type: integer
  - Valid values: x > `0`
  - Default: `100`

cell ratio
: The *cell ratio*. [*]

  - Type: null or float
  - Valid values: `null` or x > `0.0`

- Default: `null`

If `null`, the ratio is determined from the *active terminal* such that the aspect ratio of any image is always preserved. If this is not supported in the *active terminal* or on the platform, `0.5` is used instead.

cell width
The initial width of (no of columns for) grid cells, in the TUI.

- Type: integer
- Valid values: `30` <= x <= `50` and x is even
- Default: `30`

checkers
Maximum number of subprocesses for checking directory sources. [*]

- Type: null or integer
- Valid values: `null` or x >= `0`
- Default: `null`

If `null`, the number of subprocesses is automatically determined based on the amount of logical processors available. CPU affinity is also taken into account on supported platforms.
If less than `2`, directory sources are checked within the main process.

getters
Number of threads for downloading images from URL sources. [*]

- Type: integer
- Valid values: x > `0`
- Default: `4`

grid renderers
Number of subprocesses for rendering grid cells. [*]

- Type: integer
- Valid values: x >= `0`
- Default: `1`

If `0` (zero), grid cells are rendered by a thread of the main process.

log file
The file to which logs are written. [*]

- Type: string
- Valid values: An absolute path to a writable file.
- Default: `"~/.term_image/term_image.log"`

If the file doesn't exist, at least one of the parents must be a directory and be writable, so the file can created.
If the file exists, it is appended to, not overwritten.
Supports tilde expansion i.e a leading ~ (tilde) character is expanded to the current user's home directory.
See *Logging*.

> **Warning:** Relative paths are allowed but this will cause the log file to be written (or created) relative to the **current working directory** every time the process is started.

max notifications
The maximum number of TUI notifications that can be shown at a time.

- Type: integer
- Valid values: x >= `0`
- Default: `2`

Adjusts the height of the *notification bar*.

max pixels
The maximum amount of pixels in images to be displayed in the TUI. [*]

- Type: integer
- Valid values: x > `0`
- Default: `4194304` (2 ** 22)

Any image having more pixels than the specified value will be:

- skipped, in CLI mode, if `--max-pixels-cli` is specified.
- replaced, in TUI mode, with a placeholder when displayed but can still be forced to display or viewed externally.

Note that increasing this should not have any effect on general performance (i.e navigation, etc) but the larger an image is, the more the time and memory it'll take to render it. Thus, a large image might delay the rendering of other images to be rendered immediately after it.

multi
Enable or disable multiprocessing. [*]

- Type: boolean
- Valid values: `true`, `false`
- Default: `true`

If `false`, the `checkers` and `grid renderers` options have no effect.

query timeout
Timeout (in seconds) for all *Terminal Queries*. [*]

- Type: float
- Valid values: x > `0.0`
- Default: `0.1`

style
Image render style. See *Render Styles*. [*]

- Type: string
- Valid values: `"auto"`, `"block"`, `"iterm2"`, `"kitty"`
- Default: `"auto"`

If set to any value other than `"auto"` and is not overriden by the `-S | --style` command-line option, the style is used regardless of whether it's supported or not.

swap win size
A workaround for some terminal emulators (e.g older VTE-based ones) that wrongly report window dimensions swapped. [*]

- Type: boolean
- Valid values: `true`, `false`
- Default: `false`

If `true`, the dimensions reported by the terminal emulator are swapped.
This setting affects auto *Cell Ratio* computation.

4.2.2 Keybindings

The key assigned to every *action* can be modified in the config file.

The `"keys"` field in the config holds a mapping containing fields each mapping a *context* to a mapping of *actions* to their properties.

The format of the `"keys"` mapping is thus:

```
{
   "<context>": {
      "<action>": [
         "<key>",
         "<symbol>"
      ],

      ...
   },

   ...
}
```

*'...' means continuous repetition of the format **may* occur.*

action is the name of an action.
Both *key* and *symbol* must be valid Python strings, hence Unicode characters and escape sequences (`\uXXXX` and `\UXXXXXXXX`) are supported.

Hint: If using a Unicode character that occupies multiple columns in *symbol*, then add spaces after it as required to cover-up for the extra columns.

Note: The `navigation` field is not actually a *context*, instead it's the universal navigation controls configuration from which navigation *actions* in actual *contexts* are updated.

Attention: 1. Keys used in `global` context cannot be used in any other context (including `navigation`). 1. Keys used in `navigation` context cannot be used in any other context. 2. All keys in a context must be unique.

3. If a key is invalid or already used, the former and default keys for that action are tried as a fallback but if that fails (because they're already used), all keybindings from that config file are considered invalid and any changes already made are reverted.

Here is a config with Vim-style (majorly navigation) keybindings.
Remember to rename the file to `config.json` if placing it in any of the XDG Base Directories.

Below is a list of all **valid** values for *key*:

```
" "
"!"
"""
"#"
"$"
"%"
"&"
"'"
"("
")"
"*"
"+"
","
"-"
"."
"/"
"0"
"1"
"2"
"3"
"4"
"5"
"6"
"7"
"8"
"9"
":"
";"
"<"
"="
">"
"?"
"@"
"["
"\\"
"]"
"^"
"_"
```

(continues on next page)

(continued from previous page)

```
"`"
"A"
"a"
"ctrl a"
"B"
"b"
"ctrl b"
"C"
"c"
"D"
"d"
"ctrl d"
"E"
"e"
"ctrl e"
"F"
"f"
"ctrl f"
"G"
"g"
"ctrl g"
"H"
"h"
"ctrl h"
"I"
"i"
"ctrl i"
"J"
"j"
"ctrl j"
"K"
"k"
"ctrl k"
"L"
"l"
"ctrl l"
"M"
"m"
"ctrl m"
"N"
"n"
"ctrl n"
"O"
"o"
"ctrl o"
"P"
"p"
"ctrl p"
"Q"
"q"
"ctrl q"
"R"
```

(continues on next page)

(continued from previous page)

```
"r"
"ctrl r"
"S"
"s"
"ctrl s"
"T"
"t"
"ctrl t"
"U"
"u"
"ctrl u"
"V"
"v"
"ctrl v"
"W"
"w"
"ctrl w"
"X"
"x"
"ctrl x"
"Y"
"y"
"ctrl y"
"Z"
"z"
"{"
"|"
"}"
"~"
"f1"
"ctrl f1"
"shift f1"
"shift ctrl f1"
"f2"
"ctrl f2"
"shift f2"
"shift ctrl f2"
"f3"
"ctrl f3"
"shift f3"
"shift ctrl f3"
"f4"
"ctrl f4"
"shift f4"
"shift ctrl f4"
"f5"
"ctrl f5"
"shift f5"
"shift ctrl f5"
"f6"
"ctrl f6"
"shift f6"
```

(continues on next page)

(continued from previous page)

```
"shift ctrl f6"
"f7"
"ctrl f7"
"shift f7"
"shift ctrl f7"
"f8"
"ctrl f8"
"shift f8"
"shift ctrl f8"
"f9"
"ctrl f9"
"shift f9"
"shift ctrl f9"
"up"
"ctrl up"
"shift up"
"shift ctrl up"
"end"
"ctrl end"
"shift end"
"shift ctrl end"
"esc"
"f10"
"ctrl f10"
"shift f10"
"shift ctrl f10"
"f11"
"ctrl f11"
"shift f11"
"shift ctrl f11"
"f12"
"ctrl f12"
"shift f12"
"shift ctrl f12"
"tab"
"down"
"ctrl down"
"shift down"
"shift ctrl down"
"home"
"ctrl home"
"shift home"
"shift ctrl home"
"left"
"ctrl left"
"shift left"
"shift ctrl left"
"enter"
"right"
"ctrl right"
"shift right"
"shift ctrl right"
```

(continues on next page)

(continued from previous page)

```
"delete"
"ctrl delete"
"shift delete"
"shift ctrl delete"
"insert"
"backspace"
"page up"
"ctrl page up"
"page down"
"ctrl page down"
```

Any value other than these will be flagged as invalid.

The package includes a standalone in-terminal image viewer based on the library.

The image viewer is started from the command line using either the `term-image` command (only works if the Python scripts directory is on `PATH`) or `python -m term_image`.
***Take note of the differences**.

4.3 Image sources

The viewer accepts the following kinds of sources:

- A path to an image file on a local filesystem.
- A path to a directory on a local filesystem.
- An Image URL.

Any other thing given as a *source* is simply reported as invalid.

4.4 Modes

The viewer can be used in two modes:

1. **CLI mode**

   In this mode, images are directly printed to standard output. It is used when

   - output is not a terminal (even if `--tui` is specified)
   - there is only a single image source
   - the `--cli` option is specified

2. **TUI mode**

   In this mode, a Terminal/Text-based User Interface is launched, within which images and directories can be browsed and viewed in different ways. It is used when

   - there is at least one non-empty directory source
   - there are multiple image sources
   - the `--tui` option is specified

4.5 Usage

Run `term-image` with the `--help` option to see the usage info and help text.
All command-line arguments and options are described there.

Note that some options are only applicable to a specific mode. If used with the other mode, they're simply ignored.

Some options have a `[N]` (where *N* is a number) behind their description, it indicates that the option has a footnote attached.
All footnotes are at the bottom of the help text.

4.6 Render Styles

See *Render Styles*.

By default, the best style supported by the *active terminal* is automatically detected. A particular render style can be specified using the `style` *config option* or the `-S | --style` command-line option.

If the specified render style is graphics-based and not supported, an error notification is emitted and the process exits with code 1 (FAILURE, see the description below).
If the specified render style is text-based and not [fully] supported, a warning notification is emitted but execution still proceeds with the style.

The `--force-style` command-line option can be used to bypass style support checks and force the usage of any style whether it's supported or not.

4.7 Cell Ratio

The *cell ratio* is taken into consideration when setting image sizes for **text-based** render styles, in order to preserve the aspect ratio of images drawn to the terminal.

This value is determined by the *config option* `cell ratio` OR either of the command-line options `-C | --cell-ratio` or `--auto-cell-ratio`.
The command-line options are mutually exclusive and override the config option.

By default (i.e without changing the config option value or specifying either command-line option), `term-image` tries to determine the value from the *active terminal* which works on most mordern terminal emulators (currently supported on UNIX-like platforms only).
This is probably the best choice, except the terminal emulator or platform doesn't support this feature.

If `term-image` is unable to determine this value automatically, it falls back to `0.5`, which is a reasonable value in most cases.

In case *auto* cell ratio is not supported and the fallback value does not give expected results, a different value can be specified using the config or command-line option.

Attention: If using *auto* cell ratio and the *active terminal* is not the controlling terminal of the `term-image` process (e.g output is redirected to another terminal), ensure no process that might read input (e.g a shell) is currently running in the active terminal, as such a process might interfere with determining the cell ratio on some terminal emulators (e.g VTE-based ones).

For instance, the `sleep` command can be executed if a shell is currently running in the active terminal.

4.8 Notifications

Notifications are event reports meant to be brought to the immediate knowledge of the user.
Notifications have two possible destinations:

- Standard output/error stream: This is used while the TUI is **not** launched.
- TUI *notification bar*: This is used while the TUI is launched.

Notifications sent to the TUI's *notification bar* automatically disappear after 5 seconds.

4.9 Logging

Logs are more detailed event reports meant for troubleshooting and debugging purporses.

Logs are written to a file on a local filesystem. The default log file is `~/.term_image/term_image.log` but a different file can be specified:

- for all sessions, using the *log file* config option
- per session, using the `--log-file` command-line option

A log entry has the following format:

```
(<pid>) (<date> <time>) <process>: <thread>: [<level>] <module>: <function>: <message>
```

- *pid*: The process ID of the term-image session.
- *date* and *time*: Current system date and time in the format `%Y-%m-%d %H:%M:%S,<ms>`, where `<ms>` is in milliseconds.
- *process* and *thread*: The names of the python process and thread that produced the log record.
  - Only present when the *logging level* is set to `DEBUG` (either by `--debug` or `--log-level=DEBUG`).
- *level*: The level of the log entry, this indicates it's importance.
- *module*: The package sub-module from which it originated.
- *function*: The function from which it originated.
  - Only present when running on **Python 3.8+** and *logging level* is set to `DEBUG` (either by `--debug` or `--log-level=DEBUG`).
- *message*: The actual report describing the event that occured.

Note:

- Certain logs and some extra info are only provided when *logging level* is set to `DEBUG`.
- Log files are **appended to**, so it's safe use the same file for multiple sessions.
- Log files are rotated upon reaching a size of **1MiB**.
  - Only the current and immediate previous log file are kept.
- The Process ID of the `term-image` instance preceeds every log entry, so this can be used to distinguish and track logs from different sessions running simultaneously while using the same log file.

4.10 Exit Codes

`term-image` returns the following exit codes with the specified meanings:

- `0` (SUCESS): Exited normally and successfully.
- `1` (FAILURE): Exited due to an unhandled exception or a non-specific error.
- `2` (INVALID_ARG): Exited due to an invalid command-line argument value or option combination.
- `3` (INTERRUPTED): The program recieved an interrupt signal i.e `SIGINT`.
- `4` (NO_VALID_SOURCE): Exited due to lack of any valid source.

4.11 Known Issues

1. The TUI is not supported on Windows
2. Drawing of images and animations doesn't work completely well with Python for windows (tested in Windows Terminal and MinTTY). See *here* for details.

   In the viewer's CLI mode, use the `--h-allow` option to specify a horizontal allowance.
3. Some animations with the **kitty** render style within the **Kitty terminal emulator** might be glitchy at the moment. See *here* for details.

4.12 Planned Features

In no particular order:

- Performance improvements
- STDIN source
- Open image in external viewer
- Pattern-based file and directory exclusion
- Minimum and maximum file size
- Optionally following/skipping symlinks
- Distinguished color for symlinked entries in the list view

- Full grid view [TUI]
- Grid cells for directory entries [TUI]
- CLI grid view
- Interactive CLI mode
- Slideshow
- Zoom/Pan [TUI], Rotate, Flip/Mirror
- Sorting options
- Search in iist view
- Filter in list and grid views
- Alpha backaground adjustment per image
- Frame duration adjustment per animated image
- Copy:
  - Image data
  - File/Directory name
  - Full path
  - Parent directory path
- Theme customization
- Configuration menu
- Also check the library's *Planned Features* since the viewer is based on it.
- etc…

CHAPTER

FIVE

FAQS

Why?

- Why not?
- To improve and extend the capabilities of CLI and TUI applications.
- Terminals emulators have always been and always will be!

What about Windows support?

- Firstly, only the new Windows Terminal seems to have proper ANSI support and mordern terminal emulator features.
- The library and the viewer's CLI mode currently work (with a few quirks) on Windows (i.e using `cmd` or `powershell`) if the other requirements are satisfied but can't guarantee it'll always be so.
  - Drawing images and animations doesn't work completely well in `cmd` and `powershell`. See *Known Issues*.
- The TUI doesn't work due to the lack of fcntl on Windows, which is used by urwid.
- If stuck on Windows and want to use all features, you could use WSL + Windows Terminal.

Why are colours not properly reproduced?

- Some terminals support 24-bit colors but have a **256-color pallete**. This limits color reproduction.

Why are images out of scale?

- If *Auto Cell Ratio* is supported and enabled,
  - For the library, set *`SWAP_WIN_SIZE`* to `True`.
  - For the CLI or TUI, use the *swap win size* *config option* or the `--swap-win-size` command-line option.
  - If any of the above doesn't work, then open a new issue here with adequate details.
- Otherwise,
  - For the library, adjust the *cell ratio* using *`set_cell_ratio()`*.
  - For the CLI or TUI, adjust the *cell ratio* using the *config option* or the `-C | --cell-ratio` command-line option.

Why is the TUI unresponsive or slow in drawing images?

- Drawing (not rendering) speed is **entirely** dependent on the terminal emulator itself.
- Some terminal emulators block upon input, so rapidly repeated input could cause the terminal to be unresponsive.

CHAPTER

SIX

GLOSSARY

Below are definitions of terms used across the library's public interface, exception messages, CLI help text and the documentation.

Note: For contributors, these terms are also used in the source code, as variable names, in comments, docstrings, etc.

active terminal
The first terminal device discovered upon loading the package. See *here*.

alignment
The position to place a rendered image within its padding.

allowance
The amount of space to be left un-used in a given maximum size.

alpha threshold
Alpha ratio/value above which a pixel is taken as **opaque** (applies only to text-based render styles).

animated
Having multiple frames.

The frames of an animated image are generally meant to be displayed in rapid succession, to give the effect of animation.

automatic size
automatic sizing
The form of sizing wherein the image size is computed based on the *available size* or the image's original size.

See also: `Size`.

available height
The remainder after vertical allowance is subtracted from the maximum amount of lines.

available size
The remainder after *allowances* are subtracted from the maximum size.

available width
The remainder after horizontal allowance is subtracted from the maximum amount of columns.

cell ratio
The **aspect ratio** (i.e the ratio of **width to height**) of a **character cell** in the terminal emulator.

See also: `get_cell_ratio()` and `set_cell_ratio()`.

dynamic size
dynamic sizing
The form of sizing wherein the image size is automatically computed at render-time.

See also: *size*.

fixed size
fixed sizing
The form of sizing wherein the image size is set to a specific value which won't change until it is re-set.

See also: *set_size()*, *width* and *height*.

horizontal alignment
The position to place a rendered image within its *padding width*.

horizontal allowance
The amount of **columns** to be left un-used in a given maximum amount of columns.

padding
padding width
Amount of columns within which to fit an image. Excess columns on either or both sides of the image (depending on the *horizontal alignment*) will be filled with spaces.

padding height
Amount of columns within which to fit an image. Excess columns on either or both sides of the image (depending on the *vertical alignment*) will be filled with spaces.

pixel ratio
It is equvalent to the *cell ratio* multiplied by 2, since there are two pixels (arranged vertically) in one character cell.

render
rendered
rendering
To convert image pixel data into a **string** (optionally including escape sequences to produce colour and transparency).

rendered height
The amount of **lines** that'll be occupied by a rendered image **when drawn onto a terminal screen**.

rendered size
The amount of space (columns and lines) that'll be occupied by a rendered image **when drawn onto a terminal screen**.

This is determined by the size and *scale* of an image.

rendered width
The amount of **columns** that'll be occupied by a rendered image **when drawn onto a terminal screen**.

scale
The fraction of an image's size that'll actually be used to *render* it.

See also: *Image scale*.

source
The resource from which an image is derived.

terminal height
The amount of lines on a terminal screen at a time i.e without scrolling.

terminal size
The amount of columns and lines on a terminal screen at a time i.e without scrolling.

terminal width
The amount of columns on a terminal screen at a time.

vertical alignment
The position to place a rendered image within its *padding height*.

vertical allowance
: The amount of **lines** to be left un-used in a given maximum amount of lines.

CHAPTER

SEVEN

INDICES AND TABLES

- *Glossary*
- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

J

K

L

M

N

O

P

R

S

T

U

V

W